Sage Reference Manual: Algebraic Number Fields

Release 6.8

The Sage Development Team

July 29, 2015

CHAPTER

ONE

BASE CLASS FOR ALL NUMBER FIELDS

TESTS:

```
sage: k = NumberField(x^2 + 1, 'i'); k == loads(dumps(k))
True
```

class sage.rings.number_field.number_field_base.**NumberField**

Bases: sage.rings.ring.Field

Base class for all number fields.

OK(**args*, ***kwds*)

Synomym for `self.maximal_order(...)`.

EXAMPLES:

```
sage: NumberField(x^3 - 2,'a').OK()
Maximal Order in Number Field in a with defining polynomial x^3 - 2
```

bach_bound()

Return the Bach bound associated to this number field. Assuming the General Riemann Hypothesis, this is a bound B so that every integral ideal is equivalent modulo principal fractional ideals to an integral ideal of norm at most B.

See also:

`minkowski_bound()`

OUTPUT:

symbolic expression or the Integer 1

EXAMPLES:

We compute both the Minkowski and Bach bounds for a quadratic field, where the Minkowski bound is much better:

```
sage: K = QQ[sqrt(5)]
sage: K.minkowski_bound()
1/2*sqrt(5)
sage: K.minkowski_bound().n()
1.11803398874989
sage: K.bach_bound()
12*log(5)^2
sage: K.bach_bound().n()
31.0834847277628
```

We compute both the Minkowski and Bach bounds for a bigger degree field, where the Bach bound is much better:

```
sage: K = CyclotomicField(37)
sage: K.minkowski_bound().n()
7.50857335698544e14
sage: K.bach_bound().n()
191669.304126267
```

The bound of course also works for the rational numbers: sage: QQ.minkowski_bound() 1

degree()
Return the degree of this number field.

EXAMPLES:

```
sage: NumberField(x^3 + 9, 'a').degree()
3
```

discriminant()
Return the discriminant of this number field.

EXAMPLES:

```
sage: NumberField(x^3 + 9, 'a').discriminant()
-243
```

is_absolute()
Return True if self is viewed as a single extension over Q.

EXAMPLES:

```
sage: K.<a> = NumberField(x^3+2)
sage: K.is_absolute()
True
sage: y = polygen(K)
sage: L.<b> = NumberField(y^2+1)
sage: L.is_absolute()
False
sage: QQ.is_absolute()
True
```

is_finite()
Return False since number fields are not finite.

EXAMPLES:

```
sage: z = polygen(QQ)
sage: K.<theta, beta> = NumberField([z^3 - 3, z^2 + 1])
sage: K.is_finite()
False
sage: K.order()
+Infinity
```

maximal_order()
Return the maximal order, i.e., the ring of integers of this number field.

EXAMPLES:

```
sage: NumberField(x^3 - 2,'b').maximal_order()
Maximal Order in Number Field in b with defining polynomial x^3 - 2
```

minkowski_bound()

Return the Minkowski bound associated to this number field, which is a bound B so that every integral ideal is equivalent modulo principal fractional ideals to an integral ideal of norm at most B.

See also:

bach_bound()

OUTPUT:

symbolic expression or Rational

EXAMPLES:

The Minkowski bound for $\mathbf{Q}[i]$ tells us that the class number is 1:

```
sage: K = QQ[I]
sage: B = K.minkowski_bound(); B
4/pi
sage: B.n()
1.27323954473516
```

We compute the Minkowski bound for $\mathbf{Q}[\sqrt[3]{2}]$:

```
sage: K = QQ[2^(1/3)]
sage: B = K.minkowski_bound(); B
16/3*sqrt(3)/pi
sage: B.n()
2.94042077558289
sage: int(B)
2
```

We compute the Minkowski bound for $\mathbf{Q}[\sqrt{10}]$, which has class number 2:

```
sage: K = QQ[sqrt(10)]
sage: B = K.minkowski_bound(); B
sqrt(10)
sage: int(B)
3
sage: K.class_number()
2
```

We compute the Minkowski bound for $\mathbf{Q}[\sqrt{2}+\sqrt{3}]$:

```
sage: K.<y,z> = NumberField([x^2-2, x^2-3])
sage: L.<w> = QQ[sqrt(2) + sqrt(3)]
sage: B = K.minkowski_bound(); B
9/2
sage: int(B)
4
sage: B == L.minkowski_bound()
True
sage: K.class_number()
1
```

The bound of course also works for the rational numbers:

```
sage: QQ.minkowski_bound()
1
```

ring_of_integers(**args*, ***kwds*)

Synomym for `self.maximal_order(...)`.

EXAMPLES:

```
sage: K.<a> = NumberField(x^2 + 1)
sage: K.ring_of_integers()
Maximal Order in Number Field in a with defining polynomial x^2 + 1
```

signature()

Return (r1, r2), where r1 and r2 are the number of real embeddings and pairs of complex embeddings of this field, respectively.

EXAMPLES:

```
sage: NumberField(x^3 - 2, 'a').signature()
(1, 1)
```

sage.rings.number_field.number_field_base.**is_NumberField**(*x*)

Return True if x is of number field type.

EXAMPLES:

```
sage: from sage.rings.number_field.number_field_base import is_NumberField
sage: is_NumberField(NumberField(x^2+1,'a'))
True
sage: is_NumberField(QuadraticField(-97,'theta'))
True
sage: is_NumberField(CyclotomicField(97))
True
```

Note that the rational numbers QQ are a number field.:

```
sage: is_NumberField(QQ)
True
sage: is_NumberField(ZZ)
False
```

CHAPTER

TWO

NUMBER FIELDS

AUTHORS:

- William Stein (2004, 2005): initial version
- Steven Sivek (2006-05-12): added support for relative extensions
- William Stein (2007-09-04): major rewrite and documentation
- Robert Bradshaw (2008-10): specified embeddings into ambient fields
- Simon King (2010-05): Improve coercion from GAP
- Jeroen Demeyer (2010-07, 2011-04): Upgrade PARI (#9343, #10430, #11130)
- Robert Harron (2012-08): added is_CM(), complex_conjugation(), and maximal_totally_real_subfield()
- Christian Stump (2012-11): added conversion to universal cyclotomic field
- Julian Rueth (2014-04-03): absolute number fields are unique parents

---

Note: Unlike in PARI/GP, class group computations *in Sage* do *not* by default assume the Generalized Riemann Hypothesis. To do class groups computations not provably correctly you must often pass the flag `proof=False` to functions or call the function `proof.number_field(False)`. It can easily take 1000's of times longer to do computations with `proof=True` (the default).

This example follows one in the Magma reference manual:

```
sage: K.<y> = NumberField(x^4 - 420*x^2 + 40000)
sage: z = y^5/11; z
420/11*y^3 - 40000/11*y
sage: R.<y> = PolynomialRing(K)
sage: f = y^2 + y + 1
sage: L.<a> = K.extension(f); L
Number Field in a with defining polynomial y^2 + y + 1 over its base field
sage: KL.<b> = NumberField([x^4 - 420*x^2 + 40000, x^2 + x + 1]); KL
Number Field in b0 with defining polynomial x^4 - 420*x^2 + 40000 over its base field
```

We do some arithmetic in a tower of relative number fields:

```
sage: K.<cuberoot2> = NumberField(x^3 - 2)
sage: L.<cuberoot3> = K.extension(x^3 - 3)
sage: S.<sqrt2> = L.extension(x^2 - 2)
sage: S
Number Field in sqrt2 with defining polynomial x^2 - 2 over its base field
sage: sqrt2 * cuberoot3
cuberoot3*sqrt2
sage: (sqrt2 + cuberoot3)^5
```

```
(20*cuberoot3^2 + 15*cuberoot3 + 4)*sqrt2 + 3*cuberoot3^2 + 20*cuberoot3 + 60
sage: cuberoot2 + cuberoot3
cuberoot3 + cuberoot2
sage: cuberoot2 + cuberoot3 + sqrt2
sqrt2 + cuberoot3 + cuberoot2
sage: (cuberoot2 + cuberoot3 + sqrt2)^2
(2*cuberoot3 + 2*cuberoot2)*sqrt2 + cuberoot3^2 + 2*cuberoot2*cuberoot3 + cuberoot2^2 + 2
sage: cuberoot2 + sqrt2
sqrt2 + cuberoot2
sage: a = S(cuberoot2); a
cuberoot2
sage: a.parent()
Number Field in sqrt2 with defining polynomial x^2 - 2 over its base field
```

Warning: Doing arithmetic in towers of relative fields that depends on canonical coercions is currently VERY SLOW. It is much better to explicitly coerce all elements into a common field, then do arithmetic with them there (which is quite fast).

class sage.rings.number_field.number_field.**CyclotomicFieldFactory**

Bases: sage.structure.factory.UniqueFactory

Return the n-th cyclotomic field, where n is a positive integer, or the universal cyclotomic field if n==0.

For the documentation of the universal cyclotomic field, see UniversalCyclotomicField.

INPUT:

- n - a nonnegative integer, default:0
- names - name of generator (optional - defaults to zetan)
- bracket - Defines the brackets in the case of n==0, and is ignored otherwise. Can be any even length string, with "()" being the default.
- embedding - bool or n-th root of unity in an ambient field (default True)

EXAMPLES:

If called without a parameter, we get the universal cyclotomic field:

```
sage: CyclotomicField()
Universal Cyclotomic Field
```

We create the 7th cyclotomic field $\mathbf{Q}(\zeta_7)$ with the default generator name.

```
sage: k = CyclotomicField(7); k
Cyclotomic Field of order 7 and degree 6
sage: k.gen()
zeta7
```

The default embedding sends the generator to the complex primitive n^{th} root of unity of least argument.

```
sage: CC(k.gen())
0.623489801858734 + 0.781831482468030*I
```

Cyclotomic fields are of a special type.

```
sage: type(k)
<class 'sage.rings.number_field.number_field.NumberField_cyclotomic_with_category'>
```

We can specify a different generator name as follows.

```
sage: k.<z7> = CyclotomicField(7); k
Cyclotomic Field of order 7 and degree 6
sage: k.gen()
z7
```

The n must be an integer.

```
sage: CyclotomicField(3/2)
Traceback (most recent call last):
...
TypeError: no conversion of this rational to integer
```

The degree must be nonnegative.

```
sage: CyclotomicField(-1)
Traceback (most recent call last):
...
ValueError: n (=-1) must be a positive integer
```

The special case $n = 1$ does *not* return the rational numbers:

```
sage: CyclotomicField(1)
Cyclotomic Field of order 1 and degree 1
```

Due to their default embedding into $\mathbf{C}$, cyclotomic number fields are all compatible.

```
sage: cf30 = CyclotomicField(30)
sage: cf5 = CyclotomicField(5)
sage: cf3 = CyclotomicField(3)
sage: cf30.gen() + cf5.gen() + cf3.gen()
zeta30^6 + zeta30^5 + zeta30 - 1
sage: cf6 = CyclotomicField(6) ; z6 = cf6.0
sage: cf3 = CyclotomicField(3) ; z3 = cf3.0
sage: cf3(z6)
zeta3 + 1
sage: cf6(z3)
zeta6 - 1
sage: cf9 = CyclotomicField(9) ; z9 = cf9.0
sage: cf18 = CyclotomicField(18) ; z18 = cf18.0
sage: cf18(z9)
zeta18^2
sage: cf9(z18)
-zeta9^5
sage: cf18(z3)
zeta18^3 - 1
sage: cf18(z6)
zeta18^3
sage: cf18(z6)**2
zeta18^3 - 1
sage: cf9(z3)
zeta9^3
```

create_key (*n=0*, *names=None*, *embedding=True*)

Create the unique key for the cyclotomic field specified by the parameters.

TESTS:

```
sage: CyclotomicField.create_key()
(0, None, True)
```

create_object(*version*, *key*, ***extra_args*)

Create the unique cyclotomic field defined by `key`.

TESTS:

```
sage: CyclotomicField.create_object(None, (0, None, True))
Universal Cyclotomic Field
```

sage.rings.number_field.number_field.**NumberField**(*polynomial*, *name=None*, *check=True*, *names=None*, *embedding=None*, *latex_name=None*, *assume_disc_small=False*, *maximize_at_primes=None*, *structure=None*)

Return *the* number field (or tower of number fields) defined by the irreducible `polynomial`.

INPUT:

- `polynomial` - a polynomial over $\mathbf{Q}$ or a number field, or a list of such polynomials.
- `name` - a string or a list of strings, the names of the generators
- `check` - a boolean (default: `True`); do type checking and irreducibility checking.
- `embedding` - `None`, an element, or a list of elements, the images of the generators in an ambient field (default: `None`)
- `latex_name` - `None`, a string, or a list of strings (default: `None`), how the generators are printed for latex output
- `assume_disc_small` – a boolean (default: `False`); if `True`, assume that no square of a prime greater than PARI's primelimit (which should be 500000); only applies for absolute fields at present.
- `maximize_at_primes` – `None` or a list of primes (default: `None`); if not `None`, then the maximal order is computed by maximizing only at the primes in this list, which completely avoids having to factor the discriminant, but of course can lead to wrong results; only applies for absolute fields at present.
- `structure` – `None`, a list or an instance of `structure.NumberFieldStructure` (default: `None`), internally used to pass in additional structural information, e.g., about the field from which this field is created as a subfield.

EXAMPLES:

```
sage: z = QQ['z'].0
sage: K = NumberField(z^2 - 2,'s'); K
Number Field in s with defining polynomial z^2 - 2
sage: s = K.0; s
s
sage: s*s
2
sage: s^2
2
```

Constructing a relative number field:

```
sage: K.<a> = NumberField(x^2 - 2)
sage: R.<t> = K[]
sage: L.<b> = K.extension(t^3+t+a); L
Number Field in b with defining polynomial t^3 + t + a over its base field
sage: L.absolute_field('c')
Number Field in c with defining polynomial x^6 + 2*x^4 + x^2 - 2
sage: a*b
a*b
```

```
sage: L(a)
a
sage: L.lift_to_base(b^3 + b)
-a
```

Constructing another number field:

```
sage: k.<i> = NumberField(x^2 + 1)
sage: R.<z> = k[]
sage: m.<j> = NumberField(z^3 + i*z + 3)
sage: m
Number Field in j with defining polynomial z^3 + i*z + 3 over its base field
```

Number fields are globally unique:

```
sage: K.<a> = NumberField(x^3 - 5)
sage: a^3
5
sage: L.<a> = NumberField(x^3 - 5)
sage: K is L
True
```

The variable name of the defining polynomial has no influence on equality of fields:

```
sage: x = polygen(QQ, 'x'); y = polygen(QQ, 'y')
sage: k.<a> = NumberField(x^2 + 3)
sage: m.<a> = NumberField(y^2 + 3)
sage: k
Number Field in a with defining polynomial x^2 + 3
sage: m
Number Field in a with defining polynomial y^2 + 3
sage: k == m
True
```

In case of conflict of the generator name with the name given by the preparser, the name given by the preparser takes precedence:

```
sage: K.<b> = NumberField(x^2 + 5, 'a'); K
Number Field in b with defining polynomial x^2 + 5
```

One can also define number fields with specified embeddings, may be used for arithmetic and deduce relations with other number fields which would not be valid for an abstract number field.

```
sage: K.<a> = NumberField(x^3-2, embedding=1.2)
sage: RR.coerce_map_from(K)
Composite map:
  From: Number Field in a with defining polynomial x^3 - 2
  To:   Real Field with 53 bits of precision
  Defn:   Generic morphism:
          From: Number Field in a with defining polynomial x^3 - 2
          To:   Real Lazy Field
          Defn: a -> 1.259921049894873?
        then
          Conversion via _mpfr_ method map:
          From: Real Lazy Field
          To:   Real Field with 53 bits of precision
sage: RR(a)
1.25992104989487
sage: 1.1 + a
2.35992104989487
```

```
sage: b = 1/(a+1); b
1/3*a^2 - 1/3*a + 1/3
sage: RR(b)
0.442493334024442
sage: L.<b> = NumberField(x^6-2, embedding=1.1)
sage: L(a)
b^2
sage: a + b
b^2 + b
```

Note that the image only needs to be specified to enough precision to distinguish roots, and is exactly computed to any needed precision:

```
sage: RealField(200)(a)
1.2599210498948731647672106072782283505702514647015079800820
```

One can embed into any other field:

```
sage: K.<a> = NumberField(x^3-2, embedding=CC.gen()-0.6)
sage: CC(a)
-0.629960524947436 + 1.09112363597172*I
sage: L = Qp(5)
sage: f = polygen(L)^3 - 2
sage: K.<a> = NumberField(x^3-2, embedding=f.roots()[0][0])
sage: a + L(1)
4 + 2*5^2 + 2*5^3 + 3*5^4 + 5^5 + 4*5^6 + 2*5^8 + 3*5^9 + 4*5^12 + 4*5^14 + 4*5^15 + 3*5^16 + 5^
sage: L.<b> = NumberField(x^6-x^2+1/10, embedding=1)
sage: K.<a> = NumberField(x^3-x+1/10, embedding=b^2)
sage: a+b
b^2 + b
sage: CC(a) == CC(b)^2
True
sage: K.coerce_embedding()
Generic morphism:
  From: Number Field in a with defining polynomial x^3 - x + 1/10
  To:   Number Field in b with defining polynomial x^6 - x^2 + 1/10
  Defn: a -> b^2
```

The `QuadraticField` and `CyclotomicField` constructors create an embedding by default unless otherwise specified:

```
sage: K.<zeta> = CyclotomicField(15)
sage: CC(zeta)
0.913545457642601 + 0.406736643075800*I
sage: L.<sqrtn3> = QuadraticField(-3)
sage: K(sqrtn3)
2*zeta^5 + 1
sage: sqrtn3 + zeta
2*zeta^5 + zeta + 1
```

An example involving a variable name that defines a function in PARI:

```
sage: theta = polygen(QQ, 'theta')
sage: M.<z> = NumberField([theta^3 + 4, theta^2 + 3]); M
Number Field in z0 with defining polynomial theta^3 + 4 over its base field
```

TESTS:

```
sage: x = QQ['x'].gen()
sage: y = ZZ['y'].gen()
```

```
sage: K = NumberField(x^3 + x + 3, 'a'); K
Number Field in a with defining polynomial x^3 + x + 3
sage: K.defining_polynomial().parent()
Univariate Polynomial Ring in x over Rational Field

sage: L = NumberField(y^3 + y + 3, 'a'); L
Number Field in a with defining polynomial y^3 + y + 3
sage: L.defining_polynomial().parent()
Univariate Polynomial Ring in y over Rational Field

sage: W1 = NumberField(x^2+1,'a')
sage: K.<x> = CyclotomicField(5)[]
sage: W.<a> = NumberField(x^2 + 1); W
Number Field in a with defining polynomial x^2 + 1 over its base field
```

The following has been fixed in trac ticket #8800:

```
sage: P.<x> = QQ[]
sage: K.<a> = NumberField(x^3-5,embedding=0)
sage: L.<b> = K.extension(x^2+a)
sage: F, R = L.construction()
sage: F(R) == L    # indirect doctest
True
```

Check that trac ticket #11670 has been fixed:

```
sage: K.<a> = NumberField(x^2 - x - 1)
sage: loads(dumps(K)) is K
True
sage: K.<a> = NumberField(x^3 - x - 1)
sage: loads(dumps(K)) is K
True
sage: K.<a> = CyclotomicField(7)
sage: loads(dumps(K)) is K
True
```

Another problem that was found while working on trac ticket #11670, `maximize_at_primes` and `assume_disc_small` were lost when pickling:

```
sage: K.<a> = NumberField(x^3-2, assume_disc_small=True, maximize_at_primes=[2], latex_name='\\a
sage: L = loads(dumps(K))
sage: L._assume_disc_small
True
sage: L._maximize_at_primes
(2,)
```

It is an error not to specify the generator:

```
sage: K = NumberField(x^2-2)
Traceback (most recent call last):
...
TypeError: You must specify the name of the generator.
```

class sage.rings.number_field.number_field.**NumberFieldFactory**

Bases: sage.structure.factory.UniqueFactory

Factory for number fields.

This should usually not be called directly, use `NumberField()` instead.

INPUT:

- `polynomial` - a polynomial over $\mathbf{Q}$ or a number field.
- `name` - a string (default: `'a'`), the name of the generator
- `check` - a boolean (default: `True`); do type checking and irreducibility checking.
- `embedding` - `None` or an element, the images of the generator in an ambient field (default: `None`)
- `latex_name` - `None` or a string (default: `None`), how the generator is printed for latex output
- `assume_disc_small` – a boolean (default: `False`); if `True`, assume that no square of a prime greater than PARI's primelimit (which should be 500000); only applies for absolute fields at present.
- `maximize_at_primes` – `None` or a list of primes (default: `None`); if not `None`, then the maximal order is computed by maximizing only at the primes in this list, which completely avoids having to factor the discriminant, but of course can lead to wrong results; only applies for absolute fields at present.
- `structure` – `None` or an instance of `structure.NumberFieldStructure` (default: `None`), internally used to pass in additional structural information, e.g., about the field from which this field is created as a subfield.

TESTS:

```
sage: from sage.rings.number_field.number_field import NumberFieldFactory
sage: nff = NumberFieldFactory("number_field_factory")
sage: R.<x> = QQ[]
sage: nff(x^2 + 1, name='a', check=False, embedding=None, latex_name=None, assume_disc_small=Fal
Number Field in a with defining polynomial x^2 + 1
```

Pickling preserves the `structure()` of a number field:

```
sage: K.<a> = QuadraticField(2)
sage: L.<b> = K.change_names()
sage: M = loads(dumps(L))
sage: M.structure()
(Isomorphism given by variable name change map:
  From: Number Field in b with defining polynomial x^2 - 2
  To:   Number Field in a with defining polynomial x^2 - 2,
 Isomorphism given by variable name change map:
  From: Number Field in a with defining polynomial x^2 - 2
  To:   Number Field in b with defining polynomial x^2 - 2)
```

create_key_and_extra_args(*polynomial*, *name*, *check*, *embedding*, *latex_name*, *assume_disc_small*, *maximize_at_primes*, *structure*)

Create a unique key for the number field specified by the parameters.

TESTS:

```
sage: from sage.rings.number_field.number_field import NumberFieldFactory
sage: nff = NumberFieldFactory("number_field_factory")
sage: R.<x> = QQ[]
sage: nff.create_key_and_extra_args(x^2+1, name='a', check=False, embedding=None, latex_name
((Rational Field, x^2 + 1, ('a',), None, None, None, False, None), {'check': False})
```

create_object(*version*, *key*, *check*)

Create the unique number field defined by `key`.

TESTS:

```
sage: from sage.rings.number_field.number_field import NumberFieldFactory
sage: nff = NumberFieldFactory("number_field_factory")
sage: R.<x> = QQ[]
```

```
sage: nff.create_object(None, (QQ, x^2 + 1, ('a',), None, None, None, False, None), check=Fa
Number Field in a with defining polynomial x^2 + 1
```

sage.rings.number_field.number_field.**NumberFieldTower**(*polynomials*, *names*, *check=True*, *embeddings=None*, *latex_names=None*, *assume_disc_small=False*, *maximize_at_primes=None*, *structures=None*)

Create the tower of number fields defined by the polynomials in the list `polynomials`.

INPUT:

- `polynomials` - a list of polynomials. Each entry must be polynomial which is irreducible over the number field generated by the roots of the following entries.
- `names` - a list of strings or a string, the names of the generators of the relative number fields. If a single string, then names are generated from that string.
- `check` - a boolean (default: `True`), whether to check that the polynomials are irreducible
- `embeddings` - a list of elemenst or `None` (default: `None`), embeddings of the relative number fields in an ambient field.
- `latex_names` - a list of strings or `None` (default: `None`), names used to print the generators for latex output.
- `assume_disc_small` – a boolean (default: `False`); if `True`, assume that no square of a prime greater than PARI's primelimit (which should be 500000); only applies for absolute fields at present.
- `maximize_at_primes` – `None` or a list of primes (default: `None`); if not `None`, then the maximal order is computed by maximizing only at the primes in this list, which completely avoids having to factor the discriminant, but of course can lead to wrong results; only applies for absolute fields at present.
- `structures` – `None` or a list (default: `None`), internally used to provide additional information about the number field such as the field from which it was created.

OUTPUT:

Returns the relative number field generated by a root of the first entry of `polynomials` over the relative number field generated by root of the second entry of `polynomials` ... over the number field over which the last entry of `polynomials` is defined.

EXAMPLES:

```
sage: k.<a,b,c> = NumberField([x^2 + 1, x^2 + 3, x^2 + 5]); k # indirect doctest
Number Field in a with defining polynomial x^2 + 1 over its base field
sage: a^2
-1
sage: b^2
-3
sage: c^2
-5
sage: (a+b+c)^2
(2*b + 2*c)*a + 2*c*b - 9
```

The Galois group is a product of 3 groups of order 2:

```
sage: k.galois_group(type="pari")
Galois group PARI group [8, 1, 3, "E(8)=2[x]2[x]2"] of degree 8 of the Number Field in a with de
```

Repeatedly calling base_field allows us to descend the internally constructed tower of fields:

```
sage: k.base_field()
Number Field in b with defining polynomial x^2 + 3 over its base field
sage: k.base_field().base_field()
Number Field in c with defining polynomial x^2 + 5
sage: k.base_field().base_field().base_field()
Rational Field
```

In the following example the second polynomial is reducible over the first, so we get an error:

```
sage: v = NumberField([x^3 - 2, x^3 - 2], names='a')
Traceback (most recent call last):
...
ValueError: defining polynomial (x^3 - 2) must be irreducible
```

We mix polynomial parent rings:

```
sage: k.<y> = QQ[]
sage: m = NumberField([y^3 - 3, x^2 + x + 1, y^3 + 2], 'beta')
sage: m
Number Field in beta0 with defining polynomial y^3 - 3 over its base field
sage: m.base_field ()
Number Field in beta1 with defining polynomial x^2 + x + 1 over its base field
```

A tower of quadratic fields:

```
sage: K.<a> = NumberField([x^2 + 3, x^2 + 2, x^2 + 1])
sage: K
Number Field in a0 with defining polynomial x^2 + 3 over its base field
sage: K.base_field()
Number Field in a1 with defining polynomial x^2 + 2 over its base field
sage: K.base_field().base_field()
Number Field in a2 with defining polynomial x^2 + 1
```

A bigger tower of quadratic fields:

```
sage: K.<a2,a3,a5,a7> = NumberField([x^2 + p for p in [2,3,5,7]]); K
Number Field in a2 with defining polynomial x^2 + 2 over its base field
sage: a2^2
-2
sage: a3^2
-3
sage: (a2+a3+a5+a7)^3
((6*a5 + 6*a7)*a3 + 6*a7*a5 - 47)*a2 + (6*a7*a5 - 45)*a3 - 41*a5 - 37*a7
```

The function can also be called by name:

```
sage: NumberFieldTower([x^2 + 1, x^2 + 2], ['a','b'])
Number Field in a with defining polynomial x^2 + 1 over its base field
```

class sage.rings.number_field.number_field.**NumberField_absolute**(*polynomial*, *name*, *latex_name=None*, *check=True*, *embedding=None*, *assume_disc_small=False*, *maximize_at_primes=None*, *structure=None*)

Bases: sage.rings.number_field.number_field.NumberField_generic

Function to initialize an absolute number field.

EXAMPLE:

```
sage: K = NumberField(x^17 + 3, 'a'); K
Number Field in a with defining polynomial x^17 + 3
sage: type(K)
<class 'sage.rings.number_field.number_field.NumberField_absolute_with_category'>
sage: TestSuite(K).run()
```

Minkowski_embedding(*B=None*, *prec=None*)

Return an nxn matrix over RDF whose columns are the images of the basis $\{1, \alpha, \ldots, \alpha^{n-1}\}$ of self over $\mathbf{Q}$ (as vector spaces), where here α is the generator of self over $\mathbf{Q}$, i.e. self.gen(0). If B is not None, return the images of the vectors in B as the columns instead. If prec is not None, use RealField(prec) instead of RDF.

This embedding is the so-called "Minkowski embedding" of a number field in $\mathbf{R}^n$: given the n embeddings $\sigma_1, \ldots, \sigma_n$ of self in $\mathbf{C}$, write $\sigma_1, \ldots, \sigma_r$ for the real embeddings, and $\sigma_{r+1}, \ldots, \sigma_{r+s}$ for choices of one of each pair of complex conjugate embeddings (in our case, we simply choose the one where the image of α has positive real part). Here (r, s) is the signature of self. Then the Minkowski embedding is given by

$$x \mapsto (\sigma_1(x), \ldots, \sigma_r(x), \sqrt{2}\Re(\sigma_{r+1}(x)), \sqrt{2}\Im(\sigma_{r+1}(x)), \ldots, \sqrt{2}\Re(\sigma_{r+s}(x)), \sqrt{2}\Im(\sigma_{r+s}(x)))$$

Equivalently, this is an embedding of self in $\mathbf{R}^n$ so that the usual norm on $\mathbf{R}^n$ coincides with $|x| = \sum_i |\sigma_i(x)|^2$ on self.

TODO: This could be much improved by implementing homomorphisms over VectorSpaces.

EXAMPLES:

```
sage: F.<alpha> = NumberField(x^3+2)
sage: F.Minkowski_embedding()
[ 1.00000000000000 -1.25992104989487  1.58740105196820]
[ 1.41421356237... 0.8908987181... -1.12246204830...]
[0.000000000000000  1.54308184421...  1.94416129723...]
sage: F.Minkowski_embedding([1, alpha+2, alpha^2-alpha])
[ 1.00000000000000 0.740078950105127  2.84732210186307]
[ 1.41421356237...  3.7193258428... -2.01336076644...]
[0.000000000000000  1.54308184421... 0.40107945302...]
sage: F.Minkowski_embedding() * (alpha + 2).vector().column()
[0.740078950105127]
[ 3.7193258428...]
[ 1.54308184421...]
```

absolute_degree()

A synonym for degree.

EXAMPLES:

```
sage: K.<i> = NumberField(x^2 + 1)
sage: K.absolute_degree()
2
```

absolute_different()

A synonym for different.

EXAMPLES:

```
sage: K.<i> = NumberField(x^2 + 1)
sage: K.absolute_different()
Fractional ideal (2)
```

absolute_discriminant()
A synonym for discriminant.

EXAMPLES:

```
sage: K.<i> = NumberField(x^2 + 1)
sage: K.absolute_discriminant()
-4
```

absolute_generator()
An alias for sage.rings.number_field.number_field.NumberField_generic.gen(). This is provided for consistency with relative fields, where the element returned by sage.rings.number_field.number_field_rel.NumberField_relative.gen() only generates the field over its base field (not necessarily over $\mathbf{Q}$).

EXAMPLE:

```
sage: K.<a> = NumberField(x^2 - 17)
sage: K.absolute_generator()
a
```

absolute_polynomial()
A synonym for polynomial.

EXAMPLES:

```
sage: K.<i> = NumberField(x^2 + 1)
sage: K.absolute_polynomial()
x^2 + 1
```

absolute_vector_space()
A synonym for vector_space.

EXAMPLES:

```
sage: K.<i> = NumberField(x^2 + 1)
sage: K.absolute_vector_space()
(Vector space of dimension 2 over Rational Field,
 Isomorphism map:
  From: Vector space of dimension 2 over Rational Field
  To:   Number Field in i with defining polynomial x^2 + 1,
 Isomorphism map:
  From: Number Field in i with defining polynomial x^2 + 1
  To:   Vector space of dimension 2 over Rational Field)
```

automorphisms()
Compute all Galois automorphisms of self.

This uses PARI's `nfgaloisconj` and is much faster than root finding for many fields.

EXAMPLES:

```
sage: K.<a> = NumberField(x^2 + 10000)
sage: K.automorphisms()
[
Ring endomorphism of Number Field in a with defining polynomial x^2 + 10000
  Defn: a |--> a,
Ring endomorphism of Number Field in a with defining polynomial x^2 + 10000
  Defn: a |--> -a
]
```

Here's a larger example, that would take some time if we found roots instead of using PARI's specialized machinery:

```
sage: K = NumberField(x^6 - x^4 - 2*x^2 + 1, 'a')
sage: len(K.automorphisms())
2
```

L is the Galois closure of K:

```
sage: L = NumberField(x^24 - 84*x^22 + 2814*x^20 - 15880*x^18 - 409563*x^16 - 8543892*x^14 +
sage: len(L.automorphisms())
24
```

base_field()

Returns the base field of self, which is always QQ

EXAMPLES:

```
sage: K = CyclotomicField(5)
sage: K.base_field()
Rational Field
```

change_names(*names*)

Return number field isomorphic to self but with the given generator name.

INPUT:

- `names` - should be exactly one variable name.

Also, `K.structure()` returns from_K and to_K, where from_K is an isomorphism from K to self and to_K is an isomorphism from self to K.

EXAMPLES:

```
sage: K.<z> = NumberField(x^2 + 3); K
Number Field in z with defining polynomial x^2 + 3
sage: L.<ww> = K.change_names()
sage: L
Number Field in ww with defining polynomial x^2 + 3
sage: L.structure()[0]
Isomorphism given by variable name change map:
  From: Number Field in ww with defining polynomial x^2 + 3
  To:   Number Field in z with defining polynomial x^2 + 3
sage: L.structure()[0](ww + 5/3)
z + 5/3
```

elements_of_bounded_height(*bound*, *precision=53*, *LLL=False*)

Return an iterator over the elements of `self` with relative multiplicative height at most `bound`.

The algorithm requires floating point arithmetic, so the user is allowed to specify the precision for such calculations.

It might be helpful to work with an LLL-reduced system of fundamental units, so the user has the option to perform an LLL reduction for the fundamental units by setting `LLL` to True.

Certain computations may be faster assuming GRH, which may be done globally by using the number_field(True/False) switch.

For details: See [Doyle-Krumm].

INPUT:

- `bound` - a real number

- `precision` - (default: 53) a positive integer
- `LLL` - (default: False) a boolean value

OUTPUT:

- an iterator of number field elements

Warning: In the current implementation, the output of the algorithm cannot be guaranteed to be correct due to the necessity of floating point computations. In some cases, the default 53-bit precision is considerably lower than would be required for the algorithm to generate correct output.

Todo

Should implement a version of the algorithm that guarantees correct output. See Algorithm 4 in [Doyle-Krumm] for details of an implementation that takes precision issues into account.

EXAMPLES:

There are no elements in a number field with multiplicative height less than 1:

```
sage: K.<g> = NumberField(x^5 - x + 19)
sage: list(K.elements_of_bounded_height(0.9))
[]
```

The only elements in a number field of height 1 are 0 and the roots of unity:

```
sage: K.<a> = NumberField(x^2 + x + 1)
sage: list(K.elements_of_bounded_height(1))
[0, a + 1, a, -1, -a - 1, -a, 1]

sage: K.<a> = CyclotomicField(20)
sage: len(list(K.elements_of_bounded_height(1)))
21
```

The elements in the output iterator all have relative multiplicative height at most the input bound:

```
sage: K.<a> = NumberField(x^6 + 2)
sage: L = K.elements_of_bounded_height(5)
sage: for t in L:
....:     exp(6*t.global_height())
....:
1.00000000000000
1.00000000000000
1.00000000000000
2.00000000000000
2.00000000000000
2.00000000000000
2.00000000000000
4.00000000000000
4.00000000000000
4.00000000000000
4.00000000000000

sage: K.<a> = NumberField(x^2 - 71)
sage: L = K.elements_of_bounded_height(20)
sage: all(exp(2*t.global_height()) <= 20 for t in L) # long time (5 s)
True
```

```
sage: K.<a> = NumberField(x^2 + 17)
sage: L = K.elements_of_bounded_height(120)
sage: len(list(L))
9047

sage: K.<a> = NumberField(x^4 - 5)
sage: L = K.elements_of_bounded_height(50)
sage: len(list(L)) # long time (2 s)
2163

sage: K.<a> = CyclotomicField(13)
sage: L = K.elements_of_bounded_height(2)
sage: len(list(L)) # long time (3 s)
27

sage: K.<a> = NumberField(x^6 + 2)
sage: L = K.elements_of_bounded_height(60, precision=100)
sage: len(list(L)) # long time (5 s)
1899

sage: K.<a> = NumberField(x^4 - x^3 - 3*x^2 + x + 1)
sage: L = K.elements_of_bounded_height(10, LLL=true)
sage: len(list(L))
99
```

AUTHORS:

- John Doyle (2013)
- David Krumm (2013)

embeddings(*K*)

Compute all field embeddings of self into the field K (which need not even be a number field, e.g., it could be the complex numbers). This will return an identical result when given K as input again.

If possible, the most natural embedding of self into K is put first in the list.

INPUT:

- K - a number field

EXAMPLES:

```
sage: K.<a> = NumberField(x^3 - 2)
sage: L.<a1> = K.galois_closure(); L
Number Field in a1 with defining polynomial x^6 + 108
sage: K.embeddings(L)[0]
Ring morphism:
  From: Number Field in a with defining polynomial x^3 - 2
  To:   Number Field in a1 with defining polynomial x^6 + 108
  Defn: a |--> 1/18*a1^4
sage: K.embeddings(L) is K.embeddings(L)
True
```

We embed a quadratic field into a cyclotomic field:

```
sage: L.<a> = QuadraticField(-7)
sage: K = CyclotomicField(7)
sage: L.embeddings(K)
[
Ring morphism:
  From: Number Field in a with defining polynomial x^2 + 7
```

```
  To:   Cyclotomic Field of order 7 and degree 6
  Defn: a |--> 2*zeta7^4 + 2*zeta7^2 + 2*zeta7 + 1,
Ring morphism:
  From: Number Field in a with defining polynomial x^2 + 7
  To:   Cyclotomic Field of order 7 and degree 6
  Defn: a |--> -2*zeta7^4 - 2*zeta7^2 - 2*zeta7 - 1
]
```

We embed a cubic field in the complex numbers:

```
sage: K.<a> = NumberField(x^3 - 2)
sage: K.embeddings(CC)
[
Ring morphism:
  From: Number Field in a with defining polynomial x^3 - 2
  To:   Complex Field with 53 bits of precision
  Defn: a |--> -0.62996052494743... - 1.09112363597172*I,
Ring morphism:
  From: Number Field in a with defining polynomial x^3 - 2
  To:   Complex Field with 53 bits of precision
  Defn: a |--> -0.62996052494743... + 1.09112363597172*I,
Ring morphism:
  From: Number Field in a with defining polynomial x^3 - 2
  To:   Complex Field with 53 bits of precision
  Defn: a |--> 1.25992104989487
]
```

Test that trac ticket #15053 is fixed:

```
sage: K = NumberField(x^3 - 2, 'a')
sage: K.embeddings(GF(3))
[]
```

galois_closure(*names=None*, *map=False*)

Return number field K that is the Galois closure of self, i.e., is generated by all roots of the defining polynomial of self, and possibly an embedding of self into K.

INPUT:

- `names` - variable name for Galois closure
- `map` - (default: False) also return an embedding of self into K

EXAMPLES:

```
sage: K.<a> = NumberField(x^4 - 2)
sage: M = K.galois_closure('b'); M
Number Field in b with defining polynomial x^8 + 28*x^4 + 2500
sage: L.<a2> = K.galois_closure(); L
Number Field in a2 with defining polynomial x^8 + 28*x^4 + 2500
sage: K.galois_group(names=("a3")).order()
8

sage: phi = K.embeddings(L)[0]
sage: phi(K.0)
1/120*a2^5 + 19/60*a2
sage: phi(K.0).minpoly()
x^4 - 2

sage: L, phi = K.galois_closure('b', map=True)
sage: L
```

```
Number Field in b with defining polynomial x^8 + 28*x^4 + 2500
sage: phi
Ring morphism:
  From: Number Field in a with defining polynomial x^4 - 2
  To:   Number Field in b with defining polynomial x^8 + 28*x^4 + 2500
  Defn: a |--> 1/240*b^5 - 41/120*b
```

A cyclotomic field is already Galois:

```
sage: K.<a> = NumberField(cyclotomic_polynomial(23))
sage: L.<z> = K.galois_closure()
sage: L
Number Field in z with defining polynomial x^22 + x^21 + x^20 + x^19 + x^18 + x^17 + x^16 +
```

TESTS:

Let's make sure we're renaming correctly:

```
sage: K.<a> = NumberField(x^4 - 2)
sage: L, phi = K.galois_closure('cc', map=True)
sage: L
Number Field in cc with defining polynomial x^8 + 28*x^4 + 2500
sage: phi
Ring morphism:
  From: Number Field in a with defining polynomial x^4 - 2
  To:   Number Field in cc with defining polynomial x^8 + 28*x^4 + 2500
  Defn: a |--> 1/240*cc^5 - 41/120*cc
```

hilbert_conductor(a, b)

This is the product of all (finite) primes where the Hilbert symbol is -1. What is the same, this is the (reduced) discriminant of the quaternion algebra (a, b) over a number field.

INPUT:

-a,‘‘b‘‘ – elements of the number field $self$

OUTPUT:

- squarefree ideal of the ring of integers of $self$

EXAMPLES:

```
sage: F.<a> = NumberField(x^2-x-1)
sage: F.hilbert_conductor(2*a,F(-1))
Fractional ideal (2)
sage: K.<b> = NumberField(x^3-4*x+2)
sage: K.hilbert_conductor(K(2),K(-2))
Fractional ideal (1)
sage: K.hilbert_conductor(K(2*b),K(-2))
Fractional ideal (b^2 + b - 2)
```

AUTHOR:

- Aly Deines

hilbert_symbol(a, b, *P=None*)

Returns the Hilbert symbol $(a, b)_P$ for a prime P of self and non-zero elements a and b of self. If P is omitted, return the global Hilbert symbol (a, b) instead.

INPUT:

- a, b – elements of self

- P – (default: None) If P is None, compute the global symbol. Otherwise, P should be either a prime ideal of self (which may also be given as a generator or set of generators) or a real or complex embedding.

OUTPUT:

If a or b is zero, returns 0.

If a and b are non-zero and P is specified, returns the Hilbert symbol $(a, b)_P$, which is 1 if the equation $ax^2 + by^2 = 1$ has a solution in the completion of self at P, and is -1 otherwise.

If a and b are non-zero and P is unspecified, returns 1 if the equation has a solution in self and -1 otherwise.

EXAMPLES:

Some global examples:

```
sage: K.<a> = NumberField(x^2 - 23)
sage: K.hilbert_symbol(0, a+5)
0
sage: K.hilbert_symbol(a, 0)
0
sage: K.hilbert_symbol(-a, a+1)
1
sage: K.hilbert_symbol(-a, a+2)
-1
sage: K.hilbert_symbol(a, a+5)
-1
```

That the latter two are unsolvable should be visible in local obstructions. For the first, this is a prime ideal above 19. For the second, the ramified prime above 23:

```
sage: K.hilbert_symbol(-a, a+2, a+2)
-1
sage: K.hilbert_symbol(a, a+5, K.ideal(23).factor()[0][0])
-1
```

More local examples:

```
sage: K.hilbert_symbol(a, 0, K.ideal(5))
0
sage: K.hilbert_symbol(a, a+5, K.ideal(5))
1
sage: K.hilbert_symbol(a+1, 13, (a+6)*K.maximal_order())
-1
sage: [emb1, emb2] = K.embeddings(AA)
sage: K.hilbert_symbol(a, -1, emb1)
-1
sage: K.hilbert_symbol(a, -1, emb2)
1
```

Ideals P can be given by generators:

```
sage: K.<a> = NumberField(x^5 - 23)
sage: pi = 2*a^4 + 3*a^3 + 4*a^2 + 15*a + 11
sage: K.hilbert_symbol(a, a+5, pi)
1
sage: rho = 2*a^4 + 3*a^3 + 4*a^2 + 15*a + 11
sage: K.hilbert_symbol(a, a+5, rho)
1
```

This also works for non-principal ideals:

```
sage: K.<a> = QuadraticField(-5)
sage: P = K.ideal(3).factor()[0][0]
sage: P.gens_reduced()  # random, could be the other factor
(3, a + 1)
sage: K.hilbert_symbol(a, a+3, P)
1
sage: K.hilbert_symbol(a, a+3, [3, a+1])
1
```

Primes above 2:

```
sage: K.<a> = NumberField(x^5 - 23)
sage: O = K.maximal_order()
sage: p = [p[0] for p in (2*O).factor() if p[0].norm() == 16][0]
sage: K.hilbert_symbol(a, a+5, p)
1
sage: K.hilbert_symbol(a, 2, p)
1
sage: K.hilbert_symbol(-1, a-2, p)
-1
```

Various real fields are allowed:

```
sage: K.<a> = NumberField(x^3+x+1)
sage: K.hilbert_symbol(a/3, 1/2, K.embeddings(RDF)[0])
1
sage: K.hilbert_symbol(a/5, -1, K.embeddings(RR)[0])
-1
sage: [K.hilbert_symbol(a, -1, e) for e in K.embeddings(AA)]
[-1]
```

Real embeddings are not allowed to be disguised as complex embeddings:

```
sage: K.<a> = QuadraticField(5)
sage: K.hilbert_symbol(-1, -1, K.embeddings(CC)[0])
Traceback (most recent call last):
...
ValueError: Possibly real place (=Ring morphism:
  From: Number Field in a with defining polynomial x^2 - 5
  To:   Complex Field with 53 bits of precision
  Defn: a |--> -2.23606797749979) given as complex embedding in hilbert_symbol. Is it real o
sage: K.hilbert_symbol(-1, -1, K.embeddings(QQbar)[0])
Traceback (most recent call last):
...
ValueError: Possibly real place (=Ring morphism:
  From: Number Field in a with defining polynomial x^2 - 5
  To:   Algebraic Field
  Defn: a |--> -2.236067977499790?) given as complex embedding in hilbert_symbol. Is it real
sage: K.<b> = QuadraticField(-5)
sage: K.hilbert_symbol(-1, -1, K.embeddings(CDF)[0])
1
sage: K.hilbert_symbol(-1, -1, K.embeddings(QQbar)[0])
1
```

a and b do not have to be integral or coprime:

```
sage: K.<i> = QuadraticField(-1)
sage: O = K.maximal_order()
sage: K.hilbert_symbol(1/2, 1/6, 3*O)
1
```

```
sage: p = 1+i
sage: K.hilbert_symbol(p, p, p)
1
sage: K.hilbert_symbol(p, 3*p, p)
-1
sage: K.hilbert_symbol(3, p, p)
-1
sage: K.hilbert_symbol(1/3, 1/5, 1+i)
1
sage: L = QuadraticField(5, 'a')
sage: L.hilbert_symbol(-3, -1/2, 2)
1
```

Various other examples:

```
sage: K.<a> = NumberField(x^3+x+1)
sage: K.hilbert_symbol(-6912, 24, -a^2-a-2)
1
sage: K.<a> = NumberField(x^5-23)
sage: P = K.ideal(-1105*a^4 + 1541*a^3 - 795*a^2 - 2993*a + 11853)
sage: Q = K.ideal(-7*a^4 + 13*a^3 - 13*a^2 - 2*a + 50)
sage: b = -a+5
sage: K.hilbert_symbol(a,b,P)
1
sage: K.hilbert_symbol(a,b,Q)
1
sage: K.<a> = NumberField(x^5-23)
sage: P = K.ideal(-1105*a^4 + 1541*a^3 - 795*a^2 - 2993*a + 11853)
sage: K.hilbert_symbol(a, a+5, P)
1
sage: K.hilbert_symbol(a, 2, P)
1
sage: K.hilbert_symbol(a+5, 2, P)
-1
sage: K.<a> = NumberField(x^3 - 4*x + 2)
sage: K.hilbert_symbol(2, -2, K.primes_above(2)[0])
1
```

Check that the bug reported at trac ticket #16043 has been fixed:

```
sage: K.<a> = NumberField(x^2 + 5)
sage: p = K.primes_above(2)[0]; p
Fractional ideal (2, a + 1)
sage: K.hilbert_symbol(2*a, -1, p)
1
sage: K.hilbert_symbol(2*a, 2, p)
-1
sage: K.hilbert_symbol(2*a, -2, p)
-1
```

AUTHOR:

- Aly Deines (2010-08-19): part of the doctests
- Marco Streng (2010-12-06)

is_absolute()

Returns True since self is an absolute field.

EXAMPLES:

```
sage: K = CyclotomicField(5)
sage: K.is_absolute()
True
```

maximal_order(*v=None*)

Return the maximal order, i.e., the ring of integers, associated to this number field.

INPUT:

- v - (default: None) None, a prime, or a list of primes.
  - if v is None, return the maximal order.
  - if v is a prime, return an order that is p-maximal.
  - if v is a list, return an order that is maximal at each prime in the list v.

EXAMPLES:

In this example, the maximal order cannot be generated by a single element:

```
sage: k.<a> = NumberField(x^3 + x^2 - 2*x+8)
sage: o = k.maximal_order()
sage: o
Maximal Order in Number Field in a with defining polynomial x^3 + x^2 - 2*x + 8
```

We compute p-maximal orders for several p. Note that computing a p-maximal order is much faster in general than computing the maximal order:

```
sage: p = next_prime(10^22); q = next_prime(10^23)
sage: K.<a> = NumberField(x^3 - p*q)
sage: K.maximal_order([3]).basis()
[1/3*a^2 + 1/3*a + 1/3, a, a^2]
sage: K.maximal_order([2]).basis()
[1, a, a^2]
sage: K.maximal_order([p]).basis()
[1, a, a^2]
sage: K.maximal_order([q]).basis()
[1, a, a^2]
sage: K.maximal_order([p,3]).basis()
[1/3*a^2 + 1/3*a + 1/3, a, a^2]
```

An example with bigger discriminant:

```
sage: p = next_prime(10^97); q = next_prime(10^99)
sage: K.<a> = NumberField(x^3 - p*q)
sage: K.maximal_order(prime_range(10000)).basis()
[1, a, a^2]
```

optimized_representation(*names=None*, *both_maps=True*)

Return a field isomorphic to self with a better defining polynomial if possible, along with field isomorphisms from the new field to self and from self to the new field.

EXAMPLES: We construct a compositum of 3 quadratic fields, then find an optimized representation and transform elements back and forth.

```
sage: K = NumberField([x^2 + p for p in [5, 3, 2]],'a').absolute_field('b'); K
Number Field in b with defining polynomial x^8 + 40*x^6 + 352*x^4 + 960*x^2 + 576
sage: L, from_L, to_L = K.optimized_representation()
sage: L    # your answer may different, since algorithm is random
Number Field in a14 with defining polynomial x^8 + 4*x^6 + 7*x^4 + 36*x^2 + 81
sage: to_L(K.0)   # random
```

```
4/189*a14^7 - 1/63*a14^6 + 1/27*a14^5 + 2/9*a14^4 - 5/27*a14^3 + 8/9*a14^2 + 3/7*a14 + 3/7
sage: from_L(L.0)    # random
1/1152*a1^7 + 1/192*a1^6 + 23/576*a1^5 + 17/96*a1^4 + 37/72*a1^3 + 5/6*a1^2 + 55/24*a1 + 3/4
```

The transformation maps are mutually inverse isomorphisms.

```
sage: from_L(to_L(K.0))
b
sage: to_L(from_L(L.0))      # random
a14
```

optimized_subfields(*degree=0*, *name=None*, *both_maps=True*)

Return optimized representations of many (but *not* necessarily all!) subfields of self of the given degree, or of all possible degrees if degree is 0.

EXAMPLES:

```
sage: K = NumberField([x^2 + p for p in [5, 3, 2]],'a').absolute_field('b'); K
Number Field in b with defining polynomial x^8 + 40*x^6 + 352*x^4 + 960*x^2 + 576
sage: L = K.optimized_subfields(name='b')
sage: L[0][0]
Number Field in b0 with defining polynomial x - 1
sage: L[1][0]
Number Field in b1 with defining polynomial x^2 - 3*x + 3
sage: [z[0] for z in L]          # random -- since algorithm is random
[Number Field in b0 with defining polynomial x - 1,
 Number Field in b1 with defining polynomial x^2 - x + 1,
 Number Field in b2 with defining polynomial x^4 - 5*x^2 + 25,
 Number Field in b3 with defining polynomial x^4 - 2*x^2 + 4,
 Number Field in b4 with defining polynomial x^8 + 4*x^6 + 7*x^4 + 36*x^2 + 81]
```

We examine one of the optimized subfields in more detail:

```
sage: M, from_M, to_M = L[2]
sage: M                              # random
Number Field in b2 with defining polynomial x^4 - 5*x^2 + 25
sage: from_M     # may be slightly random
Ring morphism:
  From: Number Field in b2 with defining polynomial x^4 - 5*x^2 + 25
  To:   Number Field in a1 with defining polynomial x^8 + 40*x^6 + 352*x^4 + 960*x^2 + 576
  Defn: b2 |--> -5/1152*a1^7 + 1/96*a1^6 - 97/576*a1^5 + 17/48*a1^4 - 95/72*a1^3 + 17/12*a1^
```

The to_M map is None, since there is no map from K to M:

```
sage: to_M
```

We apply the from_M map to the generator of M, which gives a rather large element of K:

```
sage: from_M(M.0)          # random
-5/1152*a1^7 + 1/96*a1^6 - 97/576*a1^5 + 17/48*a1^4 - 95/72*a1^3 + 17/12*a1^2 - 53/24*a1 - 1
```

Nevertheless, that large-ish element lies in a degree 4 subfield:

```
sage: from_M(M.0).minpoly()    # random
x^4 - 5*x^2 + 25
```

order(**args*, ***kwds*)

Return the order with given ring generators in the maximal order of this number field.

INPUT:

- `gens` - list of elements in this number field; if no generators are given, just returns the cardinality of this number field (∞) for consistency.
- `check_is_integral` - bool (default: `True`), whether to check that each generator is integral.
- `check_rank` - bool (default: `True`), whether to check that the ring generated by `gens` is of full rank.
- `allow_subfield` - bool (default: `False`), if `True` and the generators do not generate an order, i.e., they generate a subring of smaller rank, instead of raising an error, return an order in a smaller number field.

EXAMPLES:

```
sage: k.<i> = NumberField(x^2 + 1)
sage: k.order(2*i)
Order in Number Field in i with defining polynomial x^2 + 1
sage: k.order(10*i)
Order in Number Field in i with defining polynomial x^2 + 1
sage: k.order(3)
Traceback (most recent call last):
...
ValueError: the rank of the span of gens is wrong
sage: k.order(i/2)
Traceback (most recent call last):
...
ValueError: each generator must be integral
```

Alternatively, an order can be constructed by adjoining elements to $\mathbf{Z}$:

```
sage: K.<a> = NumberField(x^3 - 2)
sage: ZZ[a]
Order in Number Field in a0 with defining polynomial x^3 - 2
```

TESTS:

We verify that trac ticket #2480 is fixed:

```
sage: K.<a> = NumberField(x^4 + 4*x^2 + 2)
sage: B = K.integral_basis()
sage: K.order(*B)
Order in Number Field in a with defining polynomial x^4 + 4*x^2 + 2
sage: K.order(B)
Order in Number Field in a with defining polynomial x^4 + 4*x^2 + 2
sage: K.order(gens=B)
Order in Number Field in a with defining polynomial x^4 + 4*x^2 + 2
```

places (*all_complex=False*, *prec=None*)

Return the collection of all infinite places of self.

By default, this returns the set of real places as homomorphisms into RIF first, followed by a choice of one of each pair of complex conjugate homomorphisms into CIF.

On the other hand, if prec is not None, we simply return places into RealField(prec) and ComplexField(prec) (or RDF, CDF if prec=53). One can also use `prec=infinity`, which returns embeddings into the field $\overline{\mathbf{Q}}$ of algebraic numbers (or its subfield $\mathbb{A}$ of algebraic reals); this permits exact computatation, but can be extremely slow.

There is an optional flag all_complex, which defaults to False. If all_complex is True, then the real embeddings are returned as embeddings into CIF instead of RIF.

EXAMPLES:

```
sage: F.<alpha> = NumberField(x^3-100*x+1) ; F.places()
[Ring morphism:
From: Number Field in alpha with defining polynomial x^3 - 100*x + 1
To:   Real Field with 106 bits of precision
Defn: alpha |--> -10.004996254991811845733672192 80,
Ring morphism:
From: Number Field in alpha with defining polynomial x^3 - 100*x + 1
To:   Real Field with 106 bits of precision
Defn: alpha |--> 0.010000010000030000120000550002 73,
Ring morphism:
From: Number Field in alpha with defining polynomial x^3 - 100*x + 1
To:   Real Field with 106 bits of precision
Defn: alpha |--> 9.994996244991781845613530439509]

sage: F.<alpha> = NumberField(x^3+7) ; F.places()
[Ring morphism:
From: Number Field in alpha with defining polynomial x^3 + 7
To:   Real Field with 106 bits of precision
Defn: alpha |--> -1.91293118277238910119911683954 9,
Ring morphism:
From: Number Field in alpha with defining polynomial x^3 + 7
To:   Complex Field with 53 bits of precision
Defn: alpha |--> 0.956465591386195 + 1.65664699997230*I]

sage: F.<alpha> = NumberField(x^3+7) ; F.places(all_complex=True)
[Ring morphism:
From: Number Field in alpha with defining polynomial x^3 + 7
To:   Complex Field with 53 bits of precision
Defn: alpha |--> -1.91293118277239,
Ring morphism:
From: Number Field in alpha with defining polynomial x^3 + 7
To:   Complex Field with 53 bits of precision
Defn: alpha |--> 0.956465591386195 + 1.65664699997230*I]
sage: F.places(prec=10)
[Ring morphism:
From: Number Field in alpha with defining polynomial x^3 + 7
To:   Real Field with 10 bits of precision
Defn: alpha |--> -1.9,
Ring morphism:
From: Number Field in alpha with defining polynomial x^3 + 7
To:   Complex Field with 10 bits of precision
Defn: alpha |--> 0.96 + 1.7*I]
```

real_places(*prec=None*)

Return all real places of self as homomorphisms into RIF.

EXAMPLES:

```
sage: F.<alpha> = NumberField(x^4-7) ; F.real_places()
[Ring morphism:
From: Number Field in alpha with defining polynomial x^4 - 7
To:   Real Field with 106 bits of precision
Defn: alpha |--> -1.6265765616977857432112323454 94,
Ring morphism:
From: Number Field in alpha with defining polynomial x^4 - 7
To:   Real Field with 106 bits of precision
Defn: alpha |--> 1.6265765616977857432112323454 94]
```

relative_degree()
A synonym for degree.

EXAMPLES:

```
sage: K.<i> = NumberField(x^2 + 1)
sage: K.relative_degree()
2
```

relative_different()
A synonym for different.

EXAMPLES:

```
sage: K.<i> = NumberField(x^2 + 1)
sage: K.relative_different()
Fractional ideal (2)
```

relative_discriminant()
A synonym for discriminant.

EXAMPLES:

```
sage: K.<i> = NumberField(x^2 + 1)
sage: K.relative_discriminant()
-4
```

relative_polynomial()
A synonym for polynomial.

EXAMPLES:

```
sage: K.<i> = NumberField(x^2 + 1)
sage: K.relative_polynomial()
x^2 + 1
```

relative_vector_space()
A synonym for vector_space.

EXAMPLES:

```
sage: K.<i> = NumberField(x^2 + 1)
sage: K.relative_vector_space()
(Vector space of dimension 2 over Rational Field,
 Isomorphism map:
  From: Vector space of dimension 2 over Rational Field
  To:   Number Field in i with defining polynomial x^2 + 1,
 Isomorphism map:
  From: Number Field in i with defining polynomial x^2 + 1
  To:   Vector space of dimension 2 over Rational Field)
```

relativize(*alpha*, *names*, *structure=None*)
Given an element in self or an embedding of a subfield into self, return a relative number field K isomorphic to self that is relative over the absolute field $\mathbf{Q}(\alpha)$ or the domain of $alpha$, along with isomorphisms from K to self and from self to K.

INPUT:

- `alpha` - an element of self or an embedding of a subfield into self
- `names` - 2-tuple of names of generator for output field K and the subfield QQ(alpha) names[0] generators K and names[1] QQ(alpha).

- `structure` – an instance of `structure.NumberFieldStructure` or `None` (default: `None`), if `None`, then the resulting field's `structure()` will return isomorphisms from and to this field. Otherwise, the field will be equipped with `structure`.

OUTPUT:

K – relative number field

Also, `K.structure()` returns from_K and to_K, where from_K is an isomorphism from K to self and to_K is an isomorphism from self to K.

EXAMPLES:

```
sage: K.<a> = NumberField(x^10 - 2)
sage: L.<c,d> = K.relativize(a^4 + a^2 + 2); L
Number Field in c with defining polynomial x^2 - 1/5*d^4 + 8/5*d^3 - 23/5*d^2 + 7*d - 18/5 c
sage: c.absolute_minpoly()
x^10 - 2
sage: d.absolute_minpoly()
x^5 - 10*x^4 + 40*x^3 - 90*x^2 + 110*x - 58
sage: (a^4 + a^2 + 2).minpoly()
x^5 - 10*x^4 + 40*x^3 - 90*x^2 + 110*x - 58
sage: from_L, to_L = L.structure()
sage: to_L(a)
c
sage: to_L(a^4 + a^2 + 2)
d
sage: from_L(to_L(a^4 + a^2 + 2))
a^4 + a^2 + 2
```

The following demonstrates distinct embeddings of a subfield into a larger field:

```
sage: K.<a> = NumberField(x^4 + 2*x^2 + 2)
sage: K0 = K.subfields(2)[0][0]; K0
Number Field in a0 with defining polynomial x^2 - 2*x + 2
sage: rho, tau = K0.embeddings(K)
sage: L0 = K.relativize(rho(K0.gen()), 'b'); L0
Number Field in b0 with defining polynomial x^2 - b1 + 2 over its base field
sage: L1 = K.relativize(rho, 'b'); L1
Number Field in b0 with defining polynomial x^2 - a0 + 2 over its base field
sage: L2 = K.relativize(tau, 'b'); L2
Number Field in b0 with defining polynomial x^2 + a0 over its base field
sage: L0.base_field() is K0
False
sage: L1.base_field() is K0
True
sage: L2.base_field() is K0
True
```

Here we see that with the different embeddings, the relative norms are different:

```
sage: a0 = K0.gen()
sage: L1_into_K, K_into_L1 = L1.structure()
sage: L2_into_K, K_into_L2 = L2.structure()
sage: len(K.factor(41))
4
sage: w1 = -a^2 + a + 1; P = K.ideal([w1])
sage: Pp = L1.ideal(K_into_L1(w1)).ideal_below(); Pp == K0.ideal([4*a0 + 1])
True
sage: Pp == w1.norm(rho)
True
```

```
sage: w2 = a^2 + a - 1; Q = K.ideal([w2])
sage: Qq = L2.ideal(K_into_L2(w2)).ideal_below(); Qq == K0.ideal([-4*a0 + 9])
True
sage: Qq == w2.norm(tau)
True

sage: Pp == Qq
False
```

TESTS:

We can relativize over the whole field:

```
sage: K.<a> = NumberField(x^4 + 2*x^2 + 2)
sage: K.relativize(K.gen(), 'a')
Number Field in a0 with defining polynomial x - a1 over its base field
sage: K.relativize(2*K.gen(), 'a')
Number Field in a0 with defining polynomial x - 1/2*a1 over its base field
```

We can relativize over the prime field:

```
sage: L = K.relativize(K(1), 'a'); L
Number Field in a0 with defining polynomial x^4 + 2*x^2 + 2 over its base field
sage: L.base_field()
Number Field in a1 with defining polynomial x - 1
sage: L.base_field().base_field()
Rational Field

sage: L = K.relativize(K(2), 'a'); L
Number Field in a0 with defining polynomial x^4 + 2*x^2 + 2 over its base field
sage: L.base_field()
Number Field in a1 with defining polynomial x - 2
sage: L.base_field().base_field()
Rational Field

sage: L = K.relativize(K(0), 'a'); L
Number Field in a0 with defining polynomial x^4 + 2*x^2 + 2 over its base field
sage: L.base_field()
Number Field in a1 with defining polynomial x
sage: L.base_field().base_field()
Rational Field
```

We can relativize over morphisms returned by self.subfields():

```
sage: L = NumberField(x^4 + 1, 'a')
sage: [L.relativize(h, 'c') for (f,h,i) in L.subfields()]
[Number Field in c0 with defining polynomial x^4 + 1 over its base field, Number Field in c0
```

We can relativize over a relative field:

```
sage: K.<z> = CyclotomicField(16)
sage: L, L_into_K, _ = K.subfields(4)[0]; L
Number Field in z0 with defining polynomial x^4 + 16
sage: F, F_into_L, _ = L.subfields(2)[0]; F
Number Field in z0_0 with defining polynomial x^2 + 64

sage: L_over_F = L.relativize(F_into_L, 'c'); L_over_F
Number Field in c0 with defining polynomial x^2 - 1/2*z0_0 over its base field
sage: L_over_F_into_L, _ = L_over_F.structure()
```

```
sage: K_over_rel = K.relativize(L_into_K * L_over_F_into_L, 'a'); K_over_rel
Number Field in a0 with defining polynomial x^2 - 1/2*c0 over its base field
sage: K_over_rel.base_field() is L_over_F
True
sage: K_over_rel.structure()
(Relative number field morphism:
  From: Number Field in a0 with defining polynomial x^2 - 1/2*c0 over its base field
  To:   Cyclotomic Field of order 16 and degree 8
  Defn: a0 |--> z
        c0 |--> 2*z^2
        z0_0 |--> 8*z^4, Ring morphism:
  From: Cyclotomic Field of order 16 and degree 8
  To:   Number Field in a0 with defining polynomial x^2 - 1/2*c0 over its base field
  Defn: z |--> a0)
```

We can relativize over a really large field:

```
sage: K.<a> = CyclotomicField(3^3*2^3)
sage: R = K.relativize(a^(3^2), 't'); R
Number Field in t0 with defining polynomial x^9 - t1 over its base field
sage: R.structure()
(Relative number field morphism:
  From: Number Field in t0 with defining polynomial x^9 - t1 over its base field
  To:   Cyclotomic Field of order 216 and degree 72
  Defn: t0 |--> a
        t1 |--> a^9,
 Ring morphism:
  From: Cyclotomic Field of order 216 and degree 72
  To:   Number Field in t0 with defining polynomial x^9 - t1 over its base field
  Defn: a |--> t0)
```

subfields (*degree=0*, *name=None*)

Return all subfields of self of the given degree, or of all possible degrees if degree is 0. The subfields are returned as absolute fields together with an embedding into self. For the case of the field itself, the reverse isomorphism is also provided.

EXAMPLES:

```
sage: K.<a> = NumberField( [x^3 - 2, x^2 + x + 1] )
sage: K = K.absolute_field('b')
sage: S = K.subfields()
sage: len(S)
6
sage: [k[0].polynomial() for k in S]
[x - 3,
 x^2 - 3*x + 9,
 x^3 - 3*x^2 + 3*x + 1,
 x^3 - 3*x^2 + 3*x + 1,
 x^3 - 3*x^2 + 3*x - 17,
 x^6 - 3*x^5 + 6*x^4 - 11*x^3 + 12*x^2 + 3*x + 1]
sage: R.<t> = QQ[]
sage: L = NumberField(t^3 - 3*t + 1, 'c')
sage: [k[1] for k in L.subfields()]
[Ring morphism:
  From: Number Field in c0 with defining polynomial t
  To:   Number Field in c with defining polynomial t^3 - 3*t + 1
  Defn: 0 |--> 0,
 Ring morphism:
  From: Number Field in c1 with defining polynomial t^3 - 3*t + 1
```

```
  To:   Number Field in c with defining polynomial t^3 - 3*t + 1
  Defn: c1 |--> c]
sage: len(L.subfields(2))
0
sage: len(L.subfields(1))
1
```

vector_space()

Return a vector space V and isomorphisms self –> V and V –> self.

OUTPUT:

- `V` - a vector space over the rational numbers
- `from_V` - an isomorphism from V to self
- `to_V` - an isomorphism from self to V

EXAMPLES:

```
sage: k.<a> = NumberField(x^3 + 2)
sage: V, from_V, to_V  = k.vector_space()
sage: from_V(V([1,2,3]))
3*a^2 + 2*a + 1
sage: to_V(1 + 2*a + 3*a^2)
(1, 2, 3)
sage: V
Vector space of dimension 3 over Rational Field
sage: to_V
Isomorphism map:
  From: Number Field in a with defining polynomial x^3 + 2
  To:   Vector space of dimension 3 over Rational Field
sage: from_V(to_V(2/3*a - 5/8))
2/3*a - 5/8
sage: to_V(from_V(V([0,-1/7,0])))
(0, -1/7, 0)
```

sage.rings.number_field.number_field.**NumberField_absolute_v1**(*poly*, *name*, *latex_name*, *canonical_embedding=None*)

This is used in pickling generic number fields.

EXAMPLES:

```
sage: from sage.rings.number_field.number_field import NumberField_absolute_v1
sage: R.<x> = QQ[]
sage: NumberField_absolute_v1(x^2 + 1, 'i', 'i')
Number Field in i with defining polynomial x^2 + 1
```

class sage.rings.number_field.number_field.**NumberField_cyclotomic**(*n*, *names*, *embedding=None*, *assume_disc_small=False*, *maximize_at_primes=None*)

Bases: sage.rings.number_field.number_field.NumberField_absolute

Create a cyclotomic extension of the rational field.

The command CyclotomicField(n) creates the n-th cyclotomic field, obtained by adjoining an n-th root of unity to the rational field.

EXAMPLES:

```
sage: CyclotomicField(3)
Cyclotomic Field of order 3 and degree 2
sage: CyclotomicField(18)
Cyclotomic Field of order 18 and degree 6
sage: z = CyclotomicField(6).gen(); z
zeta6
sage: z^3
-1
sage: (1+z)^3
6*zeta6 - 3

sage: K = CyclotomicField(197)
sage: loads(K.dumps()) == K
True
sage: loads((z^2).dumps()) == z^2
True

sage: cf12 = CyclotomicField( 12 )
sage: z12 = cf12.0
sage: cf6 = CyclotomicField( 6 )
sage: z6 = cf6.0
sage: FF = Frac( cf12['x'] )
sage: x = FF.0
sage: z6*x^3/(z6 + x)
zeta12^2*x^3/(x + zeta12^2)

sage: cf6 = CyclotomicField(6) ; z6 = cf6.gen(0)
sage: cf3 = CyclotomicField(3) ; z3 = cf3.gen(0)
sage: cf3(z6)
zeta3 + 1
sage: cf6(z3)
zeta6 - 1
sage: type(cf6(z3))
<type 'sage.rings.number_field.number_field_element_quadratic.NumberFieldElement_quadratic'>
sage: cf1 = CyclotomicField(1) ; z1 = cf1.0
sage: cf3(z1)
1
sage: type(cf3(z1))
<type 'sage.rings.number_field.number_field_element_quadratic.NumberFieldElement_quadratic'>
```

complex_embedding(*prec=53*)

Return the embedding of this cyclotomic field into the approximate complex field with precision prec obtained by sending the generator ζ of self to exp(2*pi*i/n), where n is the multiplicative order of ζ.

EXAMPLES:

```
sage: C = CyclotomicField(4)
sage: C.complex_embedding()
Ring morphism:
  From: Cyclotomic Field of order 4 and degree 2
  To:   Complex Field with 53 bits of precision
  Defn: zeta4 |--> 6.12323399573677e-17 + 1.00000000000000*I
```

Note in the example above that the way zeta is computed (using sin and cosine in MPFR) means that only the prec bits of the number after the decimal point are valid.

```
sage: K = CyclotomicField(3)
sage: phi = K.complex_embedding(10)
sage: phi(K.0)
-0.50 + 0.87*I
sage: phi(K.0^3)
1.0
sage: phi(K.0^3 - 1)
0.00
sage: phi(K.0^3 + 7)
8.0
```

complex_embeddings(*prec=53*)

Return all embeddings of this cyclotomic field into the approximate complex field with precision prec.

If you want 53-bit double precision, which is faster but less reliable, then do `self.embeddings(CDF)`.

EXAMPLES:

```
sage: CyclotomicField(5).complex_embeddings()
[
Ring morphism:
  From: Cyclotomic Field of order 5 and degree 4
  To:   Complex Field with 53 bits of precision
  Defn: zeta5 |--> 0.309016994374947 + 0.951056516295154*I,
Ring morphism:
  From: Cyclotomic Field of order 5 and degree 4
  To:   Complex Field with 53 bits of precision
  Defn: zeta5 |--> -0.809016994374947 + 0.587785252292473*I,
Ring morphism:
  From: Cyclotomic Field of order 5 and degree 4
  To:   Complex Field with 53 bits of precision
  Defn: zeta5 |--> -0.809016994374947 - 0.587785252292473*I,
Ring morphism:
  From: Cyclotomic Field of order 5 and degree 4
  To:   Complex Field with 53 bits of precision
  Defn: zeta5 |--> 0.309016994374947 - 0.951056516295154*I
]
```

construction()

different()

Returns the different ideal of the cyclotomic field self.

EXAMPLES:

```
sage: C20 = CyclotomicField(20)
sage: C20.different()
Fractional ideal (10, 2*zeta20^6 - 4*zeta20^4 - 4*zeta20^2 + 2)
sage: C18 = CyclotomicField(18)
sage: D = C18.different().norm()
sage: D == C18.discriminant().abs()
True
```

discriminant(*v=None*)

Returns the discriminant of the ring of integers of the cyclotomic field self, or if v is specified, the determinant of the trace pairing on the elements of the list v.

Uses the formula for the discriminant of a prime power cyclotomic field and Hilbert Theorem 88 on the discriminant of composita.

INPUT:

- v (optional) - list of element of this number field

OUTPUT: Integer if v is omitted, and Rational otherwise.

EXAMPLES:

```
sage: CyclotomicField(20).discriminant()
4000000
sage: CyclotomicField(18).discriminant()
-19683
```

is_galois()

Return True since all cyclotomic fields are automatically Galois.

EXAMPLES:

```
sage: CyclotomicField(29).is_galois()
True
```

is_isomorphic(*other*)

Return True if the cyclotomic field self is isomorphic as a number field to other.

EXAMPLES:

```
sage: CyclotomicField(11).is_isomorphic(CyclotomicField(22))
True
sage: CyclotomicField(11).is_isomorphic(CyclotomicField(23))
False
sage: CyclotomicField(3).is_isomorphic(NumberField(x^2 + x +1, 'a'))
True
sage: CyclotomicField(18).is_isomorphic(CyclotomicField(9))
True
sage: CyclotomicField(10).is_isomorphic(NumberField(x^4 - x^3 + x^2 - x + 1, 'b'))
True
```

Check trac ticket #14300:

```
sage: K = CyclotomicField(4)
sage: N = K.extension(x^2-5, 'z')
sage: K.is_isomorphic(N)
False
sage: K.is_isomorphic(CyclotomicField(8))
False
```

next_split_prime(*p=2*)

Return the next prime integer p that splits completely in this cyclotomic field (and does not ramify).

EXAMPLES:

```
sage: K.<z> = CyclotomicField(3)
sage: K.next_split_prime(7)
13
```

number_of_roots_of_unity()

Return number of roots of unity in this cyclotomic field.

EXAMPLES:

```
sage: K.<a> = CyclotomicField(21)
sage: K.number_of_roots_of_unity()
42
```

real_embeddings(*prec=53*)

Return all embeddings of this cyclotomic field into the approximate real field with precision prec.

Mostly, of course, there are no such embeddings.

EXAMPLES:

```
sage: CyclotomicField(4).real_embeddings()
[]
sage: CyclotomicField(2).real_embeddings()
[
Ring morphism:
  From: Cyclotomic Field of order 2 and degree 1
  To:   Real Field with 53 bits of precision
  Defn: -1 |--> -1.00000000000000
]
```

roots_of_unity()

Return all the roots of unity in this cyclotomic field, primitive or not.

EXAMPLES:

```
sage: K.<a> = CyclotomicField(3)
sage: zs = K.roots_of_unity(); zs
[1, a, -a - 1, -1, -a, a + 1]
sage: [ z**K.number_of_roots_of_unity() for z in zs ]
[1, 1, 1, 1, 1, 1]
```

signature()

Return (r1, r2), where r1 and r2 are the number of real embeddings and pairs of complex embeddings of this cyclotomic field, respectively.

Trivial since, apart from QQ, cyclotomic fields are totally complex.

EXAMPLES:

```
sage: CyclotomicField(5).signature()
(0, 2)
sage: CyclotomicField(2).signature()
(1, 0)
```

zeta(*n=None*, *all=False*)

Returns an element of multiplicative order n in this this cyclotomic field, if there is one. Raises a ValueError if there is not.

INPUT:

- `n` - integer (default: None, returns element of maximal order)
- `all` - bool (default: False) - whether to return a list of all n-th roots.

OUTPUT: root of unity or list

EXAMPLES:

```
sage: k = CyclotomicField(7)
sage: k.zeta()
zeta7
sage: k.zeta().multiplicative_order()
7
sage: k = CyclotomicField(49)
sage: k.zeta().multiplicative_order()
49
```

```
sage: k.zeta(7).multiplicative_order()
7
sage: k.zeta()
zeta49
sage: k.zeta(7)
zeta49^7

sage: K.<a> = CyclotomicField(7)
sage: K.zeta(14, all=True)
[-a^4, -a^5, a^5 + a^4 + a^3 + a^2 + a + 1, -a, -a^2, -a^3]
sage: K.<a> = CyclotomicField(10)
sage: K.zeta(20, all=True)
Traceback (most recent call last):
...
ValueError: n (=20) does not divide order of generator

sage: K.<a> = CyclotomicField(5)
sage: K.zeta(4)
Traceback (most recent call last):
...
ValueError: n (=4) does not divide order of generator
sage: v = K.zeta(5, all=True); v
[a, a^2, a^3, -a^3 - a^2 - a - 1]
sage: [b^5 for b in v]
[1, 1, 1, 1]
```

zeta_order()

Return the order of the maximal root of unity contained in this cyclotomic field.

EXAMPLES:

```
sage: CyclotomicField(1).zeta_order()
2
sage: CyclotomicField(4).zeta_order()
4
sage: CyclotomicField(5).zeta_order()
10
sage: CyclotomicField(5)._n()
5
sage: CyclotomicField(389).zeta_order()
778
```

sage.rings.number_field.number_field.**NumberField_cyclotomic_v1**(*zeta_order*, *name*, *canonical_embedding=None*)

This is used in pickling cyclotomic fields.

EXAMPLES:

```
sage: from sage.rings.number_field.number_field import NumberField_cyclotomic_v1
sage: NumberField_cyclotomic_v1(5,'a')
Cyclotomic Field of order 5 and degree 4
sage: NumberField_cyclotomic_v1(5,'a').variable_name()
'a'
```

class sage.rings.number_field.number_field.**NumberField_generic**(*polynomial*, *name*, *latex_name=None*, *check=True*, *embedding=None*, *category=None*, *assume_disc_small=False*, *maximize_at_primes=None*, *structure=None*)

Bases: sage.rings.number_field.number_field_base.NumberField

Generic class for number fields defined by an irreducible polynomial over $\mathbf{Q}$.

EXAMPLES:

```
sage: K.<a> = NumberField(x^3 - 2); K
Number Field in a with defining polynomial x^3 - 2
sage: TestSuite(K).run()
```

S_class_group(*S*, *proof=None*, *names='c'*)

Returns the S-class group of this number field over its base field.

INPUT:

- `S` - a set of primes of the base field
- `proof` - if False, assume the GRH in computing the class group. Default is True. Call `number_field_proof` to change this default globally.
- `names` - names of the generators of this class group.

OUTPUT:

The S-class group of this number field.

EXAMPLE:

A well known example:

```
sage: K.<a> = QuadraticField(-5)
sage: K.S_class_group([])
S-class group of order 2 with structure C2 of Number Field in a with defining polynomial x^2
```

When we include the prime $(2, a + 1)$, the S-class group becomes trivial:

```
sage: K.S_class_group([K.ideal(2,a+1)])
S-class group of order 1 of Number Field in a with defining polynomial x^2 + 5
```

TESTS:

```
sage: K.<a> = QuadraticField(-14)
sage: I = K.ideal(2,a)
sage: S = (I,)
sage: CS = K.S_class_group(S);CS
S-class group of order 2 with structure C2 of Number Field in a with defining polynomial x^2
sage: T = tuple([])
sage: CT = K.S_class_group(T);CT
S-class group of order 4 with structure C4 of Number Field in a with defining polynomial x^2
sage: K.class_group()
Class group of order 4 with structure C4 of Number Field in a with defining polynomial x^2 +
```

S_unit_group(*proof=None*, *S=None*)

Return the S-unit group (including torsion) of this number field.

ALGORITHM: Uses PARI's bnfsunit command.

INPUT:

- `proof` (bool, default True) flag passed to `pari`.
- `S` - list or tuple of prime ideals, or an ideal, or a single ideal or element from which an ideal can be constructed, in which case the support is used. If None, the global unit group is constructed; otherwise, the S-unit group is constructed.

Note: The group is cached.

EXAMPLES:

```
sage: x = polygen(QQ)
sage: K.<a> = NumberField(x^4 - 10*x^3 + 20*5*x^2 - 15*5^2*x + 11*5^3)
sage: U = K.S_unit_group(S=a); U
S-unit group with structure C10 x Z x Z x Z of Number Field in a with defining polynomial x^
sage: U.gens()
(u0, u1, u2, u3)
sage: U.gens_values()
[-7/275*a^3 + 1/11*a^2 - 9/11*a - 1, 7/275*a^3 - 1/11*a^2 + 9/11*a + 2, 1/275*a^3 + 4/55*a^2
sage: U.invariants()
(10, 0, 0, 0)
sage: [u.multiplicative_order() for u in U.gens()]
[10, +Infinity, +Infinity, +Infinity]
sage: U.primes()
(Fractional ideal (5, 1/275*a^3 + 4/55*a^2 - 5/11*a + 5), Fractional ideal (11, 1/275*a^3 +
```

With the default value of S, the S-unit group is the same as the global unit group:

```
sage: x = polygen(QQ)
sage: K.<a> = NumberField(x^3 + 3)
sage: U = K.unit_group(proof=False)
sage: U == K.S_unit_group(proof=False)
True
```

The value of S may be specified as a list of prime ideals, or an ideal, or an element of the field:

```
sage: K.<a> = NumberField(x^3 + 3)
sage: U = K.S_unit_group(proof=False, S=K.ideal(6).prime_factors()); U
S-unit group with structure C2 x Z x Z x Z x Z of Number Field in a with defining polynomial
sage: K.<a> = NumberField(x^3 + 3)
sage: U = K.S_unit_group(proof=False, S=K.ideal(6)); U
S-unit group with structure C2 x Z x Z x Z x Z of Number Field in a with defining polynomial
sage: K.<a> = NumberField(x^3 + 3)
sage: U = K.S_unit_group(proof=False, S=6); U
S-unit group with structure C2 x Z x Z x Z x Z of Number Field in a with defining polynomial

sage: U
S-unit group with structure C2 x Z x Z x Z x Z of Number Field in a with defining polynomial
sage: U.primes()
(Fractional ideal (-a^2 + a - 1),
Fractional ideal (a + 1),
Fractional ideal (a))
sage: U.gens()
(u0, u1, u2, u3, u4)
sage: U.gens_values()
[-1, a^2 - 2, -a^2 + a - 1, a + 1, a]
```

The exp and log methods can be used to create S-units from sequences of exponents, and recover the exponents:

```
sage: U.gens_orders()
(2, 0, 0, 0, 0)
sage: u = U.exp((3,1,4,1,5)); u
-6*a^2 + 18*a - 54
sage: u.norm().factor()
-1 * 2^9 * 3^5
sage: U.log(u)
(1, 1, 4, 1, 5)
```

S_units(*S*, *proof=True*)

Returns a list of generators of the S-units.

INPUT:

- `S` – a set of primes of the base field
- `proof` – if `False`, assume the GRH in computing the class group

OUTPUT:

A list of generators of the unit group.

Note:

For more functionality see the S_unit_group() function.

EXAMPLE:

```
sage: K.<a> = QuadraticField(-3)
sage: K.unit_group()
Unit group with structure C6 of Number Field in a with defining polynomial x^2 + 3
sage: K.S_units([])
[-1/2*a + 1/2]
sage: K.S_units([])[0].multiplicative_order()
6
```

An example in a relative extension (see trac ticket #8722):

```
sage: L.<a,b> = NumberField([x^2 + 1, x^2 - 5])
sage: p = L.ideal((-1/2*b - 1/2)*a + 1/2*b - 1/2)
sage: W = L.S_units([p]); [x.norm() for x in W]
[9, 1, 1]
```

Our generators should have the correct parent (trac ticket #9367):

```
sage: _.<x> = QQ[]
sage: L.<alpha> = NumberField(x^3 + x + 1)
sage: p = L.S_units([ L.ideal(7) ])
sage: p[0].parent()
Number Field in alpha with defining polynomial x^3 + x + 1
```

TEST:

This checks that the multiple entries issue at trac ticket #9341 is fixed:

```
sage: _.<t> = QQ[]
sage: K.<T> = NumberField(t-1)
sage: I = K.ideal(2)
sage: K.S_units([I])
[2, -1]
```

```
sage: J = K.ideal(-2)
sage: K.S_units([I, J, I])
[2, -1]
```

---

absolute_degree()
Return the degree of self over $\mathbf{Q}$.

EXAMPLES:

```
sage: NumberField(x^3 + x^2 + 997*x + 1, 'a').absolute_degree()
3
sage: NumberField(x + 1, 'a').absolute_degree()
1
sage: NumberField(x^997 + 17*x + 3, 'a', check=False).absolute_degree()
997
```

absolute_field(*names*)
Returns self as an absolute extension over QQ.

OUTPUT:

- K - this number field (since it is already absolute)

Also, `K.structure()` returns from_K and to_K, where from_K is an isomorphism from K to self and to_K is an isomorphism from self to K.

EXAMPLES:

```
sage: K = CyclotomicField(5)
sage: K.absolute_field('a')
Number Field in a with defining polynomial x^4 + x^3 + x^2 + x + 1
```

absolute_polynomial_ntl()
Alias for `polynomial_ntl()`. Mostly for internal use.

EXAMPLES:

```
sage: NumberField(x^2 + (2/3)*x - 9/17,'a').absolute_polynomial_ntl()
([-27 34 51], 51)
```

algebraic_closure()
Return the algebraic closure of self (which is QQbar).

EXAMPLES:

```
sage: K.<i> = QuadraticField(-1)
sage: K.algebraic_closure()
Algebraic Field
sage: K.<a> = NumberField(x^3-2)
sage: K.algebraic_closure()
Algebraic Field
sage: K = CyclotomicField(23)
sage: K.algebraic_closure()
Algebraic Field
```

change_generator(*alpha*, *name=None*, *names=None*)
Given the number field self, construct another isomorphic number field K generated by the element alpha of self, along with isomorphisms from K to self and from self to K.

EXAMPLES:

```
sage: L.<i> = NumberField(x^2 + 1); L
Number Field in i with defining polynomial x^2 + 1
sage: K, from_K, to_K = L.change_generator(i/2 + 3)
sage: K
Number Field in i0 with defining polynomial x^2 - 6*x + 37/4
sage: from_K
Ring morphism:
  From: Number Field in i0 with defining polynomial x^2 - 6*x + 37/4
  To:   Number Field in i with defining polynomial x^2 + 1
  Defn: i0 |--> 1/2*i + 3
sage: to_K
Ring morphism:
  From: Number Field in i with defining polynomial x^2 + 1
  To:   Number Field in i0 with defining polynomial x^2 - 6*x + 37/4
  Defn: i |--> 2*i0 - 6
```

We can also do

```
sage: K.<c>, from_K, to_K = L.change_generator(i/2 + 3); K
Number Field in c with defining polynomial x^2 - 6*x + 37/4
```

We compute the image of the generator $\sqrt{-1}$ of L.

```
sage: to_K(i)
2*c - 6
```

Note that the image is indeed a square root of -1.

```
sage: to_K(i)^2
-1
sage: from_K(to_K(i))
i
sage: to_K(from_K(c))
c
```

characteristic()

Return the characteristic of this number field, which is of course 0.

EXAMPLES:

```
sage: k.<a> = NumberField(x^99 + 2); k
Number Field in a with defining polynomial x^99 + 2
sage: k.characteristic()
0
```

class_group(*proof=None*, *names='c'*)

Return the class group of the ring of integers of this number field.

INPUT:

- `proof` - if True then compute the class group provably correctly. Default is True. Call number_field_proof to change this default globally.
- `names` - names of the generators of this class group.

OUTPUT: The class group of this number field.

EXAMPLES:

```
sage: K.<a> = NumberField(x^2 + 23)
sage: G = K.class_group(); G
Class group of order 3 with structure C3 of Number Field in a with defining polynomial x^2 +
```

```
sage: G.0
Fractional ideal class (2, 1/2*a - 1/2)
sage: G.gens()
(Fractional ideal class (2, 1/2*a - 1/2),)

sage: G.number_field()
Number Field in a with defining polynomial x^2 + 23
sage: G is K.class_group()
True
sage: G is K.class_group(proof=False)
False
sage: G.gens()
(Fractional ideal class (2, 1/2*a - 1/2),)
```

There can be multiple generators:

```
sage: k.<a> = NumberField(x^2 + 20072)
sage: G = k.class_group(); G
Class group of order 76 with structure C38 x C2 of Number Field in a with defining polynomia
sage: G.0 # random
Fractional ideal class (41, a + 10)
sage: G.0^38
Trivial principal fractional ideal class
sage: G.1 # random
Fractional ideal class (2, -1/2*a)
sage: G.1^2
Trivial principal fractional ideal class
```

Class groups of Hecke polynomials tend to be very small:

```
sage: f = ModularForms(97, 2).T(2).charpoly()
sage: f.factor()
(x - 3) * (x^3 + 4*x^2 + 3*x - 1) * (x^4 - 3*x^3 - x^2 + 6*x - 1)
sage: [NumberField(g,'a').class_group().order() for g,_ in f.factor()]
[1, 1, 1]
```

class_number (*proof=None*)

Return the class number of this number field, as an integer.

INPUT:

- `proof` - bool (default: True unless you called number_field_proof)

EXAMPLES:

```
sage: NumberField(x^2 + 23, 'a').class_number()
3
sage: NumberField(x^2 + 163, 'a').class_number()
1
sage: NumberField(x^3 + x^2 + 997*x + 1, 'a').class_number(proof=False)
1539
```

completion (*p*, *prec*, *extras={}*)

Returns the completion of self at p to the specified precision. Only implemented at archimedean places, and then only if an embedding has been fixed.

EXAMPLES:

```
sage: K.<a> = QuadraticField(2)
sage: K.completion(infinity, 100)
Real Field with 100 bits of precision
```

```
sage: K.<zeta> = CyclotomicField(12)
sage: K.completion(infinity, 53, extras={'type': 'RDF'})
Complex Double Field
sage: zeta + 1.5                              # implicit test
2.36602540378444 + 0.500000000000000*I
```

complex_conjugation()

Return the complex conjugation of self.

This is only well-defined for fields contained in CM fields (i.e. for totally real fields and CM fields). Recall that a CM field is a totally imaginary quadratic extension of a totally real field. For other fields, a ValueError is raised.

EXAMPLES:

```
sage: QuadraticField(-1, 'I').complex_conjugation()
Ring endomorphism of Number Field in I with defining polynomial x^2 + 1
  Defn: I |--> -I
sage: CyclotomicField(8).complex_conjugation()
Ring endomorphism of Cyclotomic Field of order 8 and degree 4
  Defn: zeta8 |--> -zeta8^3
sage: QuadraticField(5, 'a').complex_conjugation()
Identity endomorphism of Number Field in a with defining polynomial x^2 - 5
sage: F = NumberField(x^4 + x^3 - 3*x^2 - x + 1, 'a')
sage: F.is_totally_real()
True
sage: F.complex_conjugation()
Identity endomorphism of Number Field in a with defining polynomial x^4 + x^3 - 3*x^2 - x +
sage: F.<b> = NumberField(x^2 - 2)
sage: F.extension(x^2 + 1, 'a').complex_conjugation()
Relative number field endomorphism of Number Field in a with defining polynomial x^2 + 1 ove
  Defn: a |--> -a
        b |--> b
sage: F2.<b> = NumberField(x^2 + 2)
sage: K2.<a> = F2.extension(x^2 + 1)
sage: cc = K2.complex_conjugation()
sage: cc(a)
-a
sage: cc(b)
-b
```

complex_embeddings(*prec=53*)

Return all homomorphisms of this number field into the approximate complex field with precision prec.

This always embeds into an MPFR based complex field. If you want embeddings into the 53-bit double precision, which is faster, use `self.embeddings(CDF)`.

EXAMPLES:

```
sage: k.<a> = NumberField(x^5 + x + 17)
sage: v = k.complex_embeddings()
sage: ls = [phi(k.0^2) for phi in v] ; ls # random order
[2.97572074038...,
 -2.40889943716 + 1.90254105304*I,
 -2.40889943716 - 1.90254105304*I,
 0.921039066973 + 3.07553311885*I,
 0.921039066973 - 3.07553311885*I]
sage: K.<a> = NumberField(x^3 + 2)
sage: ls = K.complex_embeddings() ; ls # random order
[
```

```
Ring morphism:
  From: Number Field in a with defining polynomial x^3 + 2
  To:   Complex Double Field
  Defn: a |--> -1.25992104989...,
Ring morphism:
  From: Number Field in a with defining polynomial x^3 + 2
  To:   Complex Double Field
  Defn: a |--> 0.629960524947 - 1.09112363597*I,
Ring morphism:
  From: Number Field in a with defining polynomial x^3 + 2
  To:   Complex Double Field
  Defn: a |--> 0.629960524947 + 1.09112363597*I
]
```

composite_fields(*other*, *names=None*, *both_maps=False*, *preserve_embedding=True*)

List of all possible composite number fields formed from self and other, together with (optionally) embeddings into the compositum; see the documentation for both_maps below.

If preserve_embedding is True and if self and other both have embeddings into the same ambient field, or into fields which are contained in a common field, only the compositum respecting both embeddings is returned. If one (or both) of self or other does not have an embedding or preserve_embedding is False, all possible composite number fields are returned.

INPUT:

- `other` - a number field
- `names` - generator name for composite fields
- `both_maps` - (default: False) if True, return quadruples (F, self_into_F, other_into_F, k) such that self_into_F is an embedding of self in F, other_into_F is an embedding of in F, and k is an integer such that F.gen() equals other_into_F(other.gen()) + k*self_into_F(self.gen()) or has the value Infinity in which case F.gen() equals self_into_F(self.gen()), or is None (which happens when other is a relative number field). If both self and other have embeddings into an ambient field, then F will have an embedding with respect to which both self_into_F and other_into_F will be compatible with the ambient embeddings.
- `preserve_embedding` - (default: True) if self and other have ambient embeddings, then return only the compatible compositum.

OUTPUT:

- `list` - list of the composite fields, possibly with maps.

EXAMPLES:

```
sage: K.<a> = NumberField(x^4 - 2)
sage: K.composite_fields(K)
[Number Field in a with defining polynomial x^4 - 2,
 Number Field in a0 with defining polynomial x^8 + 28*x^4 + 2500]
```

A particular compositum is selected, together with compatible maps into the compositum, if the fields are endowed with a real or complex embedding:

```
sage: K1 = NumberField(x^4 - 2, 'a', embedding=RR(2^(1/4)))
sage: K2 = NumberField(x^4 - 2, 'a', embedding=RR(-2^(1/4)))
sage: K1.composite_fields(K2)
[Number Field in a with defining polynomial x^4 - 2]
sage: [F, f, g, k], = K1.composite_fields(K2, both_maps=True); F
Number Field in a with defining polynomial x^4 - 2
```

```
sage: f(K1.0), g(K2.0)
(a, -a)
```

With preserve_embedding set to False, the embeddings are ignored:

```
sage: K1.composite_fields(K2, preserve_embedding=False)
[Number Field in a with defining polynomial x^4 - 2,
 Number Field in a0 with defining polynomial x^8 + 28*x^4 + 2500]
```

Changing the embedding selects a different compositum:

```
sage: K3 = NumberField(x^4 - 2, 'a', embedding=CC(2^(1/4)*I))
sage: [F, f, g, k], = K1.composite_fields(K3, both_maps=True); F
Number Field in a0 with defining polynomial x^8 + 28*x^4 + 2500
sage: f(K1.0), g(K3.0)
(1/240*a0^5 - 41/120*a0, 1/120*a0^5 + 19/60*a0)
```

If no embeddings are specified, the maps into the composite are chosen arbitrarily:

```
sage: Q1.<a> = NumberField(x^4 + 10*x^2 + 1)
sage: Q2.<b> = NumberField(x^4 + 16*x^2 + 4)
sage: Q1.composite_fields(Q2, 'c')
[Number Field in c with defining polynomial x^8 + 64*x^6 + 904*x^4 + 3840*x^2 + 3600]
sage: F, Q1_into_F, Q2_into_F, k = Q1.composite_fields(Q2, 'c', both_maps=True)[0]
sage: Q1_into_F
Ring morphism:
  From: Number Field in a with defining polynomial x^4 + 10*x^2 + 1
  To:   Number Field in c with defining polynomial x^8 + 64*x^6 + 904*x^4 + 3840*x^2 + 3600
  Defn: a |--> 19/14400*c^7 + 137/1800*c^5 + 2599/3600*c^3 + 8/15*c
```

This is just one of four embeddings of Q1 into F:: sage: Hom(Q1, F).order() 4

TESTS:

Let's check that embeddings are being respected:

```
sage: x = polygen(ZZ)
sage: K0.<b> = CyclotomicField(7, 'a').subfields(3)[0][0].change_names()
sage: K1.<a1> = K0.extension(x^2 - 2*b^2, 'a1').absolute_field()
sage: K2.<a2> = K0.extension(x^2 - 3*b^2, 'a2').absolute_field()
```

We need embeddings, so we redefine:

```
sage: L1.<a1> = NumberField(K1.polynomial(), 'a1', embedding=CC.0)
sage: L2.<a2> = NumberField(K2.polynomial(), 'a2', embedding=CC.0)
sage: [CDF(a1), CDF(a2)]
[-0.6293842454258951, -0.7708351267200304]
```

and we get the same embeddings via the compositum:

```
sage: F, L1_into_F, L2_into_F, k = L1.composite_fields(L2, both_maps=True)[0]
sage: [CDF(L1_into_F(L1.gen())), CDF(L2_into_F(L2.gen()))]
[-0.6293842454258959, -0.7708351267200312]
```

Let's check that if only one field has an embedding, the resulting fields do not have embeddings:

```
sage: L1.composite_fields(K2)[0].coerce_embedding() is None
True
sage: L2.composite_fields(K1)[0].coerce_embedding() is None
True
```

We check that other can be a relative number field:

```
sage: L.<a, b> = NumberField([x^3 - 5, x^2 + 3])
sage: CyclotomicField(3, 'w').composite_fields(L, both_maps=True)
[(Number Field in a with defining polynomial x^3 - 5 over its base field, Ring morphism:
  From: Cyclotomic Field of order 3 and degree 2
  To:   Number Field in a with defining polynomial x^3 - 5 over its base field
  Defn: w |--> -1/2*b - 1/2, Relative number field endomorphism of Number Field in a with de
  Defn: a |--> a
        b |--> b, None)]
```

construction()

Construction of self

EXAMPLE:

```
sage: K.<a>=NumberField(x^3+x^2+1,embedding=CC.gen())
sage: F,R = K.construction()
sage: F
AlgebraicExtensionFunctor
sage: R
Rational Field
sage: F(R) == K
True
```

Note that, if a number field is provided with an embedding, the construction functor applied to the rationals is not necessarily identic with the number field. The reason is that the construction functor uses a value for the embedding that is equivalent, but not necessarily equal, to the one provided in the definition of the number field:

```
sage: F(R) is K
False
sage: F.embeddings
[0.2327856159383841? + 0.7925519925154479?*I]
```

TEST:

```
sage: K.<a> = NumberField(x^3+x+1)
sage: R.<t> = ZZ[]
sage: a+t      # indirect doctest
t + a
sage: (a+t).parent()
Univariate Polynomial Ring in t over Number Field in a with defining polynomial x^3 + x + 1
```

The construction works for non-absolute number fields as well:

```
sage: K.<a,b,c>=NumberField([x^3+x^2+1,x^2+1,x^7+x+1])
sage: F,R = K.construction()
sage: F(R) == K
True

sage: P.<x> = QQ[]
sage: K.<a> = NumberField(x^3-5,embedding=0)
sage: L.<b> = K.extension(x^2+a)
sage: a*b
a*b
```

defining_polynomial()

Return the defining polynomial of this number field.

This is exactly the same as `self.polynomal()`.

EXAMPLES:

```
sage: k5.<z> = CyclotomicField(5)
sage: k5.defining_polynomial()
x^4 + x^3 + x^2 + x + 1
sage: y = polygen(QQ,'y')
sage: k.<a> = NumberField(y^9 - 3*y + 5); k
Number Field in a with defining polynomial y^9 - 3*y + 5
sage: k.defining_polynomial()
y^9 - 3*y + 5
```

degree()

Return the degree of this number field.

EXAMPLES:

```
sage: NumberField(x^3 + x^2 + 997*x + 1, 'a').degree()
3
sage: NumberField(x + 1, 'a').degree()
1
sage: NumberField(x^997 + 17*x + 3, 'a', check=False).degree()
997
```

different()

Compute the different fractional ideal of this number field.

The codifferent is the fractional ideal of all x in K such that the the trace of xy is an integer for all $y \in O_K$.

The different is the integral ideal which is the inverse of the codifferent.

EXAMPLES:

```
sage: k.<a> = NumberField(x^2 + 23)
sage: d = k.different()
sage: d
Fractional ideal (-a)
sage: d.norm()
23
sage: k.disc()
-23
```

The different is cached:

```
sage: d is k.different()
True
```

Another example:

```
sage: k.<b> = NumberField(x^2 - 123)
sage: d = k.different(); d
Fractional ideal (2*b)
sage: d.norm()
492
sage: k.disc()
492
```

disc(*v=None*)

Shortcut for self.discriminant.

EXAMPLES:

```
sage: k.<b> = NumberField(x^2 - 123)
sage: k.disc()
```

```
492
```

discriminant(*v=None*)

Returns the discriminant of the ring of integers of the number field, or if v is specified, the determinant of the trace pairing on the elements of the list v.

INPUT:

- `v (optional)` - list of element of this number field

OUTPUT: Integer if v is omitted, and Rational otherwise.

EXAMPLES:

```
sage: K.<t> = NumberField(x^3 + x^2 - 2*x + 8)
sage: K.disc()
-503
sage: K.disc([1, t, t^2])
-2012
sage: K.disc([1/7, (1/5)*t, (1/3)*t^2])
-2012/11025
sage: (5*7*3)^2
11025
sage: NumberField(x^2 - 1/2, 'a').discriminant()
8
```

elements_of_norm(*n*, *proof=None*)

Return a list of solutions modulo units of positive norm to $Norm(a) = n$, where a can be any integer in this number field.

INPUT:

- `proof` - default: True, unless you called number_field_proof and set it otherwise.

EXAMPLES:

```
sage: K.<a> = NumberField(x^2+1)
sage: K.elements_of_norm(3)
[]
sage: K.elements_of_norm(50)
[-7*a + 1, -5*a - 5, 7*a + 1]
```

extension(*poly*, *name=None*, *names=None*, *check=True*, *embedding=None*, *latex_name=None*, *structure=None*)

Return the relative extension of this field by a given polynomial.

EXAMPLES:

```
sage: K.<a> = NumberField(x^3 - 2)
sage: R.<t> = K[]
sage: L.<b> = K.extension(t^2 + a); L
Number Field in b with defining polynomial t^2 + a over its base field
```

We create another extension:

```
sage: k.<a> = NumberField(x^2 + 1); k
Number Field in a with defining polynomial x^2 + 1
sage: y = var('y')
sage: m.<b> = k.extension(y^2 + 2); m
Number Field in b with defining polynomial y^2 + 2 over its base field
```

Note that b is a root of $y^2 + 2$:

```
sage: b.minpoly()
x^2 + 2
sage: b.minpoly('z')
z^2 + 2
```

A relative extension of a relative extension:

```
sage: k.<a> = NumberField([x^2 + 1, x^3 + x + 1])
sage: R.<z> = k[]
sage: L.<b> = NumberField(z^3 + 3 + a); L
Number Field in b with defining polynomial z^3 + a0 + 3 over its base field
```

factor(n)

Ideal factorization of the principal ideal generated by n.

EXAMPLES:

Here we show how to factor Gaussian integers (up to units). First we form a number field defined by $x^2 + 1$:

```
sage: K.<I> = NumberField(x^2 + 1); K
Number Field in I with defining polynomial x^2 + 1
```

Here are the factors:

```
sage: fi, fj = K.factor(17); fi,fj
((Fractional ideal (I + 4), 1), (Fractional ideal (I - 4), 1))
```

Now we extract the reduced form of the generators:

```
sage: zi = fi[0].gens_reduced()[0]; zi
I + 4
sage: zj = fj[0].gens_reduced()[0]; zj
I - 4
```

We recover the integer that was factored in $\mathbf{Z}[i]$ (up to a unit):

```
sage: zi*zj
-17
```

One can also factor elements or ideals of the number field:

```
sage: K.<a> = NumberField(x^2 + 1)
sage: K.factor(1/3)
(Fractional ideal (3))^-1
sage: K.factor(1+a)
Fractional ideal (a + 1)
sage: K.factor(1+a/5)
(Fractional ideal (a + 1)) * (Fractional ideal (-a - 2))^-1 * (Fractional ideal (2*a + 1))^-
```

An example over a relative number field:

```
sage: pari('setrand(2)')
sage: L.<b> = K.extension(x^2 - 7)
sage: f = L.factor(a + 1); f
(Fractional ideal (1/2*a*b - a + 1/2)) * (Fractional ideal (-1/2*a*b - a + 1/2))
sage: f.value() == a+1
True
```

It doesn't make sense to factor the ideal (0), so this raises an error:

```
sage: L.factor(0)
Traceback (most recent call last):
...
AttributeError: 'NumberFieldIdeal' object has no attribute 'factor'
```

AUTHORS:

- Alex Clemesha (2006-05-20), Francis Clarke (2009-04-21): examples

fractional_ideal(**gens*, ***kwds*)

Return the ideal in $\mathcal{O}_K$ generated by gens. This overrides the `sage.rings.ring.Field` method to use the `sage.rings.ring.Ring` one instead, since we're not really concerned with ideals in a field but in its ring of integers.

INPUT:

- `gens` - a list of generators, or a number field ideal.

EXAMPLES:

```
sage: K.<a> = NumberField(x^3-2)
sage: K.fractional_ideal([1/a])
Fractional ideal (1/2*a^2)
```

One can also input a number field ideal itself, or, more usefully, for a tower of number fields an ideal in one of the fields lower down the tower.

```
sage: K.fractional_ideal(K.ideal(a))
Fractional ideal (a)
sage: L.<b> = K.extension(x^2 - 3, x^2 + 1)
sage: M.<c> = L.extension(x^2 + 1)
sage: L.ideal(K.ideal(2, a))
Fractional ideal (a)
sage: M.ideal(K.ideal(2, a)) == M.ideal(a*(b - c)/2)
True
```

The zero ideal is not a fractional ideal!

```
sage: K.fractional_ideal(0)
Traceback (most recent call last):
...
ValueError: gens must have a nonzero element (zero ideal is not a fractional ideal)
```

galois_group(*type=None*, *algorithm='pari'*, *names=None*)

Return the Galois group of the Galois closure of this number field.

INPUT:

- `type` - `none`, `gap`, or `pari`. If None (the default), return an explicit group of automorphisms of self as a `GaloisGroup_v2` object. Otherwise, return a `GaloisGroup_v1` wrapper object based on a PARI or Gap transitive group object, which is quicker to compute, but rather less useful (in particular, it can't be made to act on self). If type = 'gap', the database_gap package should be installed.
- `algorithm` - 'pari', 'kash', 'magma'. (default: 'pari', except when the degree is >= 12 when 'kash' is tried.)
- `name` - a string giving a name for the generator of the Galois closure of self, when self is not Galois. This is ignored if type is not None.

Note that computing Galois groups as abstract groups is often much faster than computing them as explicit automorphism groups (but of course you get less information out!) For more (impor-

tant!) documentation, so the documentation for Galois groups of polynomials over $\mathbf{Q}$, e.g., by typing `K.polynomial().galois_group?`, where K is a number field.

To obtain actual field homomorphisms from the number field to its splitting field, use type=None.

EXAMPLES:

With type `None`:

```
sage: k.<b> = NumberField(x^2 - 14) # a Galois extension
sage: G = k.galois_group(); G
Galois group of Number Field in b with defining polynomial x^2 - 14
sage: G.gen(0)
(1,2)
sage: G.gen(0)(b)
-b
sage: G.artin_symbol(k.primes_above(3)[0])
(1,2)

sage: k.<b> = NumberField(x^3 - x + 1) # not Galois
sage: G = k.galois_group(names='c'); G
Galois group of Galois closure in c of Number Field in b with defining polynomial x^3 - x +
sage: G.gen(0)
(1,2,3)(4,5,6)
```

With type `'pari'`:

```
sage: NumberField(x^3-2, 'a').galois_group(type="pari")
Galois group PARI group [6, -1, 2, "S3"] of degree 3 of the Number Field in a with defining

sage: NumberField(x-1, 'a').galois_group(type="gap")    # optional - database_gap
Galois group Transitive group number 1 of degree 1 of the Number Field in a with defining po
sage: NumberField(x^2+2, 'a').galois_group(type="gap")  # optional - database_gap
Galois group Transitive group number 1 of degree 2 of the Number Field in a with defining po
sage: NumberField(x^3-2, 'a').galois_group(type="gap")  # optional - database_gap
Galois group Transitive group number 2 of degree 3 of the Number Field in a with defining po

sage: x = polygen(QQ)
sage: NumberField(x^3 + 2*x + 1, 'a').galois_group(type='gap')    # optional - database_gap
Galois group Transitive group number 2 of degree 3 of the Number Field in a with defining po
sage: NumberField(x^3 + 2*x + 1, 'a').galois_group(algorithm='magma')   # opticnal - magma d
Galois group Transitive group number 2 of degree 3 of the Number Field in a with defining po
```

EXPLICIT GALOIS GROUP: We compute the Galois group as an explicit group of automorphisms of the Galois closure of a field.

```
sage: K.<a> = NumberField(x^3 - 2)
sage: L.<b1> = K.galois_closure(); L
Number Field in b1 with defining polynomial x^6 + 108
sage: G = End(L); G
Automorphism group of Number Field in b1 with defining polynomial x^6 + 108
sage: G.list()
[
Ring endomorphism of Number Field in b1 with defining polynomial x^6 + 108
  Defn: b1 |--> b1,
...
Ring endomorphism of Number Field in b1 with defining polynomial x^6 + 108
  Defn: b1 |--> -1/12*b1^4 - 1/2*b1
]
sage: G[2](b1)
1/12*b1^4 + 1/2*b1
```

gen(*n=0*)

Return the generator for this number field.

INPUT:

- n - must be 0 (the default), or an exception is raised.

EXAMPLES:

```
sage: k.<theta> = NumberField(x^14 + 2); k
Number Field in theta with defining polynomial x^14 + 2
sage: k.gen()
theta
sage: k.gen(1)
Traceback (most recent call last):
...
IndexError: Only one generator.
```

gen_embedding()

If an embedding has been specified, return the image of the generator under that embedding. Otherwise return None.

EXAMPLES:

```
sage: QuadraticField(-7, 'a').gen_embedding()
2.645751311064591?*I
sage: NumberField(x^2+7, 'a').gen_embedding() # None
```

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

K.ideal() returns a fractional ideal of the field, except for the zero ideal which is not a fractional ideal.

EXAMPLES:

```
sage: K.<i>=NumberField(x^2+1)
sage: K.ideal(2)
Fractional ideal (2)
sage: K.ideal(2+i)
Fractional ideal (i + 2)
sage: K.ideal(0)
Ideal (0) of Number Field in i with defining polynomial x^2 + 1
```

ideals_of_bdd_norm(*bound*)

All integral ideals of bounded norm.

INPUT:

- bound - a positive integer

OUTPUT: A dict of all integral ideals I such that Norm(I) <= bound, keyed by norm.

EXAMPLE:

```
sage: K.<a> = NumberField(x^2 + 23)
sage: d = K.ideals_of_bdd_norm(10)
sage: for n in d:
....:     print n
....:     for I in d[n]:
....:         print I
1
Fractional ideal (1)
2
```

```
Fractional ideal (2, 1/2*a - 1/2)
Fractional ideal (2, 1/2*a + 1/2)
3
Fractional ideal (3, 1/2*a - 1/2)
Fractional ideal (3, 1/2*a + 1/2)
4
Fractional ideal (4, 1/2*a + 3/2)
Fractional ideal (2)
Fractional ideal (4, 1/2*a + 5/2)
5
6
Fractional ideal (1/2*a - 1/2)
Fractional ideal (6, 1/2*a + 5/2)
Fractional ideal (6, 1/2*a + 7/2)
Fractional ideal (1/2*a + 1/2)
7
8
Fractional ideal (1/2*a + 3/2)
Fractional ideal (4, a - 1)
Fractional ideal (4, a + 1)
Fractional ideal (1/2*a - 3/2)
9
Fractional ideal (9, 1/2*a + 11/2)
Fractional ideal (3)
Fractional ideal (9, 1/2*a + 7/2)
10
```

integral_basis(*v=None*)

Returns a list containing a ZZ-basis for the full ring of integers of this number field.

INPUT:

- v - None, a prime, or a list of primes. See the documentation for self.maximal_order.

EXAMPLES:

```
sage: K.<a> = NumberField(x^5 + 10*x + 1)
sage: K.integral_basis()
[1, a, a^2, a^3, a^4]
```

Next we compute the ring of integers of a cubic field in which 2 is an "essential discriminant divisor", so the ring of integers is not generated by a single element.

```
sage: K.<a> = NumberField(x^3 + x^2 - 2*x + 8)
sage: K.integral_basis()
[1, 1/2*a^2 + 1/2*a, a^2]
```

ALGORITHM: Uses the pari library (via _pari_integral_basis).

is_CM()

Return True if self is a CM field (i.e. a totally imaginary quadratic extension of a totally real field).

EXAMPLES:

```
sage: Q.<a> = NumberField(x - 1)
sage: Q.is_CM()
False
sage: K.<i> = NumberField(x^2 + 1)
sage: K.is_CM()
True
sage: L.<zeta20> = CyclotomicField(20)
```

```
sage: L.is_CM()
True
sage: K.<omega> = QuadraticField(-3)
sage: K.is_CM()
True
sage: L.<sqrt5> = QuadraticField(5)
sage: L.is_CM()
False
sage: F.<a> = NumberField(x^3 - 2)
sage: F.is_CM()
False
sage: F.<a> = NumberField(x^4-x^3-3*x^2+x+1)
sage: F.is_CM()
False
```

The following are non-CM totally imaginary fields.

```
sage: F.<a> = NumberField(x^4 + x^3 - x^2 - x + 1)
sage: F.is_totally_imaginary()
True
sage: F.is_CM()
False
sage: F2.<a> = NumberField(x^12 - 5*x^11 + 8*x^10 - 5*x^9 - \
                           x^8 + 9*x^7 + 7*x^6 - 3*x^5 + 5*x^4 + \
                           7*x^3 - 4*x^2 - 7*x + 7)
sage: F2.is_totally_imaginary()
True
sage: F2.is_CM()
False
```

The following is a non-cyclotomic CM field.

```
sage: M.<a> = NumberField(x^4 - x^3 - x^2 - 2*x + 4)
sage: M.is_CM()
True
```

Now, we construct a totally imaginary quadratic extension of a totally real field (which is not cyclotomic).

```
sage: E_0.<a> = NumberField(x^7 - 4*x^6 - 4*x^5 + 10*x^4 + 4*x^3 - \
                            6*x^2 - x + 1)
sage: E_0.is_totally_real()
True
sage: E.<b> = E_0.extension(x^2 + 1)
sage: E.is_CM()
True
```

Finally, a CM field that is given as an extension that is not CM.

```
sage: E_0.<a> = NumberField(x^2 - 4*x + 16)
sage: y = polygen(E_0)
sage: E.<z> = E_0.extension(y^2 - E_0.gen() / 2)
sage: E.is_CM()
True
sage: E.is_CM_extension()
False
```

is_absolute()

Returns True if self is an absolute field.

This function will be implemented in the derived classes.

EXAMPLES:

```
sage: K = CyclotomicField(5)
sage: K.is_absolute()
True
```

is_field(*proof=True*)

Return True since a number field is a field.

EXAMPLES:

```
sage: NumberField(x^5 + x + 3, 'c').is_field()
True
```

is_galois()

Return True if this number field is a Galois extension of $\mathbf{Q}$.

EXAMPLES:

```
sage: NumberField(x^2 + 1, 'i').is_galois()
True
sage: NumberField(x^3 + 2, 'a').is_galois()
False
```

is_isomorphic(*other*)

Return True if self is isomorphic as a number field to other.

EXAMPLES:

```
sage: k.<a> = NumberField(x^2 + 1)
sage: m.<b> = NumberField(x^2 + 4)
sage: k.is_isomorphic(m)
True
sage: m.<b> = NumberField(x^2 + 5)
sage: k.is_isomorphic (m)
False

sage: k = NumberField(x^3 + 2, 'a')
sage: k.is_isomorphic(NumberField((x+1/3)^3 + 2, 'b'))
True
sage: k.is_isomorphic(NumberField(x^3 + 4, 'b'))
True
sage: k.is_isomorphic(NumberField(x^3 + 5, 'b'))
False
```

is_relative()

EXAMPLES:

```
sage: K.<a> = NumberField(x^10 - 2)
sage: K.is_absolute()
True
sage: K.is_relative()
False
```

is_totally_imaginary()

Return True if self is totally imaginary, and False otherwise.

Totally imaginary means that no isomorphic embedding of self into the complex numbers has image contained in the real numbers.

EXAMPLES:

```
sage: NumberField(x^2+2, 'alpha').is_totally_imaginary()
True
sage: NumberField(x^2-2, 'alpha').is_totally_imaginary()
False
sage: NumberField(x^4-2, 'alpha').is_totally_imaginary()
False
```

is_totally_real()

Return True if self is totally real, and False otherwise.

Totally real means that every isomorphic embedding of self into the complex numbers has image contained in the real numbers.

EXAMPLES:

```
sage: NumberField(x^2+2, 'alpha').is_totally_real()
False
sage: NumberField(x^2-2, 'alpha').is_totally_real()
True
sage: NumberField(x^4-2, 'alpha').is_totally_real()
False
```

latex_variable_name(*name=None*)

Return the latex representation of the variable name for this number field.

EXAMPLES:

```
sage: NumberField(x^2 + 3, 'a').latex_variable_name()
'a'
sage: NumberField(x^3 + 3, 'theta3').latex_variable_name()
'\theta_{3}'
sage: CyclotomicField(5).latex_variable_name()
'\zeta_{5}'
```

maximal_totally_real_subfield()

Return the maximal totally real subfield of self together with an embedding of it into self.

EXAMPLES:

```
sage: F.<a> = QuadraticField(11)
sage: F.maximal_totally_real_subfield()
[Number Field in a with defining polynomial x^2 - 11, Identity endomorphism of Number Field
sage: F.<a> = QuadraticField(-15)
sage: F.maximal_totally_real_subfield()
[Rational Field, Natural morphism:
  From: Rational Field
  To:   Number Field in a with defining polynomial x^2 + 15]
sage: F.<a> = CyclotomicField(29)
sage: F.maximal_totally_real_subfield()
(Number Field in a0 with defining polynomial x^14 + x^13 - 13*x^12 - 12*x^11 + 66*x^10 + 55*
  From: Number Field in a0 with defining polynomial x^14 + x^13 - 13*x^12 - 12*x^11 + 66*x^1
  To:   Cyclotomic Field of order 29 and degree 28
  Defn: a0 |--> -a^27 - a^26 - a^25 - a^24 - a^23 - a^22 - a^21 - a^20 - a^19 - a^18 - a^17
sage: F.<a> = NumberField(x^3 - 2)
sage: F.maximal_totally_real_subfield()
[Rational Field, Conversion map:
  From: Rational Field
  To:   Number Field in a with defining polynomial x^3 - 2]
sage: F.<a> = NumberField(x^4 - x^3 - x^2 + x + 1)
sage: F.maximal_totally_real_subfield()
```

```
[Rational Field, Conversion map:
  From: Rational Field
  To:   Number Field in a with defining polynomial x^4 - x^3 - x^2 + x + 1]
sage: F.<a> = NumberField(x^4 - x^3 + 2*x^2 + x + 1)
sage: F.maximal_totally_real_subfield()
[Number Field in a1 with defining polynomial x^2 - x - 1, Ring morphism:
  From: Number Field in a1 with defining polynomial x^2 - x - 1
  To:   Number Field in a with defining polynomial x^4 - x^3 + 2*x^2 + x + 1
  Defn: a1 |--> -1/2*a^3 - 1/2]
sage: F.<a> = NumberField(x^4-4*x^2-x+1)
sage: F.maximal_totally_real_subfield()
[Number Field in a with defining polynomial x^4 - 4*x^2 - x + 1, Identity endomorphism of Nu
```

An example of a relative extension where the base field is not the maximal totally real subfield.

```
sage: E_0.<a> = NumberField(x^2 - 4*x + 16)
sage: y = polygen(E_0)
sage: E.<z> = E_0.extension(y^2 - E_0.gen() / 2)
sage: E.maximal_totally_real_subfield()
[Number Field in z2 with defining polynomial x^2 - 6, Composite map:
  From: Number Field in z2 with defining polynomial x^2 - 6
  To:   Number Field in z with defining polynomial x^2 - 1/2*a over its base field
  Defn:   Ring morphism:
          From: Number Field in z2 with defining polynomial x^2 - 6
          To:   Number Field in z with defining polynomial x^4 - 2*x^2 + 4
          Defn: z2 |--> -1/2*z^3 + 2*z
        then
          Isomorphism map:
          From: Number Field in z with defining polynomial x^4 - 2*x^2 + 4
          To:   Number Field in z with defining polynomial x^2 - 1/2*a over its base field]
```

narrow_class_group(*proof=None*)

Return the narrow class group of this field.

INPUT:

- `proof` - default: None (use the global proof setting, which defaults to True).

EXAMPLES:

```
sage: NumberField(x^3+x+9, 'a').narrow_class_group()
Multiplicative Abelian group isomorphic to C2
```

ngens()

Return the number of generators of this number field (always 1).

OUTPUT: the python integer 1.

EXAMPLES:

```
sage: NumberField(x^2 + 17,'a').ngens()
1
sage: NumberField(x + 3,'a').ngens()
1
sage: k.<a> = NumberField(x + 3)
sage: k.ngens()
1
sage: k.0
-3
```

number_of_roots_of_unity()
Return the number of roots of unity in this field.

Note: We do not create the full unit group since that can be expensive, but we do use it if it is already known.

EXAMPLES:

```
sage: F.<alpha> = NumberField(x**22+3)
sage: F.zeta_order()
6
sage: F.<alpha> = NumberField(x**2-7)
sage: F.zeta_order()
2
```

order()
Return the order of this number field (always +infinity).

OUTPUT: always positive infinity

EXAMPLES:

```
sage: NumberField(x^2 + 19,'a').order()
+Infinity
```

pari_bnf(*proof=None*, *units=True*)
PARI big number field corresponding to this field.

INPUT:

- `proof` – If False, assume GRH. If True, run PARI's `bnfcertify()` to make sure that the results are correct.
- `units` – (default: True) If True, insist on having fundamental units. If False, the units may or may not be computed.

OUTPUT:

The PARI `bnf` structure of this number field.

Warning: Even with `proof=True`, I wouldn't trust this to mean that everything computed involving this number field is actually correct.

EXAMPLES:

```
sage: k.<a> = NumberField(x^2 + 1); k
Number Field in a with defining polynomial x^2 + 1
sage: len(k.pari_bnf())
10
sage: k.pari_bnf()[:4]
[[;], matrix(0,3), [;], ...]
sage: len(k.pari_nf())
9
sage: k.<a> = NumberField(x^7 + 7); k
Number Field in a with defining polynomial x^7 + 7
sage: dummy = k.pari_bnf(proof=True)
```

pari_nf(*important=True*)
PARI number field corresponding to this field.

This is the number field constructed using nfinit(). This is the same as the number field got by doing pari(self) or gp(self).

INPUT:

- `important` – (default: True) bool. If False, raise a `RuntimeError` if we need to do a difficult discriminant factorization. Useful when the PARI nf structure is useful but not strictly required, such as for factoring polynomials over this number field.

EXAMPLES:

```
sage: k.<a> = NumberField(x^4 - 3*x + 7); k
Number Field in a with defining polynomial x^4 - 3*x + 7
sage: k.pari_nf()[:4]
[y^4 - 3*y + 7, [0, 2], 85621, 1]
sage: pari(k)[:4]
[y^4 - 3*y + 7, [0, 2], 85621, 1]

sage: k.<a> = NumberField(x^4 - 3/2*x + 5/3); k
Number Field in a with defining polynomial x^4 - 3/2*x + 5/3
sage: k.pari_nf()
Traceback (most recent call last):
...
TypeError: Unable to coerce number field defined by non-integral polynomial to PARI.
sage: pari(k)
Traceback (most recent call last):
...
TypeError: Unable to coerce number field defined by non-integral polynomial to PARI.
sage: gp(k)
Traceback (most recent call last):
...
TypeError: Unable to coerce number field defined by non-integral polynomial to PARI.
```

With `important=False`, we simply bail out if we cannot easily factor the discriminant:

```
sage: p = next_prime(10^40); q = next_prime(10^41)
sage: K.<a> = NumberField(x^2 - p*q)
sage: K.pari_nf(important=False)
Traceback (most recent call last):
...
RuntimeError: Unable to factor discriminant with trial division
```

Next, we illustrate the `maximize_at_primes` and `assume_disc_small` parameters of the NumberField contructor. The following would take a very long time without the `maximize_at_primes` option:

```
sage: K.<a> = NumberField(x^2 - p*q, maximize_at_primes=[p])
sage: K.pari_nf()
[y^2 - 100000000000000000000...]
```

Since the discriminant is square-free, this also works:

```
sage: K.<a> = NumberField(x^2 - p*q, assume_disc_small=True)
sage: K.pari_nf()
[y^2 - 100000000000000000000...]
```

pari_polynomial (*name='x'*)

PARI polynomial with integer coefficients corresponding to the polynomial that defines this number field.

By default, this is a polynomial in the variable "x". PARI prefers integral polynomials, so we clear the denominator. Therefore, this is NOT the same as simply converting the defining polynomial to PARI.

EXAMPLES:

```
sage: y = polygen(QQ)
sage: k.<a> = NumberField(y^2 - 3/2*y + 5/3)
sage: k.pari_polynomial()
6*x^2 - 9*x + 10
sage: k.polynomial()._pari_()
x^2 - 3/2*x + 5/3
sage: k.pari_polynomial('a')
6*a^2 - 9*a + 10
```

This fails with arguments which are not a valid PARI variable name:

```
sage: k = QuadraticField(-1)
sage: k.pari_polynomial('I')
Traceback (most recent call last):
...
PariError: I already exists with incompatible valence
sage: k.pari_polynomial('i')
i^2 + 1
sage: k.pari_polynomial('theta')
Traceback (most recent call last):
...
PariError: theta already exists with incompatible valence
```

pari_rnfnorm_data(*L*, *proof=True*)

Return the PARI rnfisnorminit() data corresponding to the extension L/self.

EXAMPLES:

```
sage: x = polygen(QQ)
sage: K = NumberField(x^2 - 2, 'alpha')
sage: L = K.extension(x^2 + 5, 'gamma')
sage: ls = K.pari_rnfnorm_data(L) ; len(ls)
8

sage: K.<a> = NumberField(x^2 + x + 1)
sage: P.<X> = K[]
sage: L.<b> = NumberField(X^3 + a)
sage: ls = K.pari_rnfnorm_data(L); len(ls)
8
```

pari_zk()

Integral basis of the PARI number field corresponding to this field.

This is the same as pari_nf().getattr('zk'), but much faster.

EXAMPLES:

```
sage: k.<a> = NumberField(x^3 - 17)
sage: k.pari_zk()
[1, 1/3*y^2 - 1/3*y + 1/3, y]
sage: k.pari_nf().getattr('zk')
[1, 1/3*y^2 - 1/3*y + 1/3, y]
```

polynomial()

Return the defining polynomial of this number field.

This is exactly the same as `self.defining_polynomal()`.

EXAMPLES:

```
sage: NumberField(x^2 + (2/3)*x - 9/17,'a').polynomial()
x^2 + 2/3*x - 9/17
```

polynomial_ntl()

Return defining polynomial of this number field as a pair, an ntl polynomial and a denominator.

This is used mainly to implement some internal arithmetic.

EXAMPLES:

```
sage: NumberField(x^2 + (2/3)*x - 9/17,'a').polynomial_ntl()
([-27 34 51], 51)
```

polynomial_quotient_ring()

Return the polynomial quotient ring isomorphic to this number field.

EXAMPLES:

```
sage: K = NumberField(x^3 + 2*x - 5, 'alpha')
sage: K.polynomial_quotient_ring()
Univariate Quotient Polynomial Ring in alpha over Rational Field with modulus x^3 + 2*x - 5
```

polynomial_ring()

Return the polynomial ring that we view this number field as being a quotient of (by a principal ideal).

EXAMPLES: An example with an absolute field:

```
sage: k.<a> = NumberField(x^2 + 3)
sage: y = polygen(QQ, 'y')
sage: k.<a> = NumberField(y^2 + 3)
sage: k.polynomial_ring()
Univariate Polynomial Ring in y over Rational Field
```

An example with a relative field:

```
sage: y = polygen(QQ, 'y')
sage: M.<a> = NumberField([y^3 + 97, y^2 + 1]); M
Number Field in a0 with defining polynomial y^3 + 97 over its base field
sage: M.polynomial_ring()
Univariate Polynomial Ring in y over Number Field in a1 with defining polynomial y^2 + 1
```

power_basis()

Return a power basis for this number field over its base field.

If this number field is represented as $k[t]/f(t)$, then the basis returned is $1, t, t^2, \ldots, t^{d-1}$ where d is the degree of this number field over its base field.

EXAMPLES:

```
sage: K.<a> = NumberField(x^5 + 10*x + 1)
sage: K.power_basis()
[1, a, a^2, a^3, a^4]

sage: L.<b> = K.extension(x^2 - 2)
sage: L.power_basis()
[1, b]
sage: L.absolute_field('c').power_basis()
[1, c, c^2, c^3, c^4, c^5, c^6, c^7, c^8, c^9]

sage: M = CyclotomicField(15)
sage: M.power_basis()
[1, zeta15, zeta15^2, zeta15^3, zeta15^4, zeta15^5, zeta15^6, zeta15^7]
```

prime_above(*x*, *degree=None*)

Return a prime ideal of self lying over x.

INPUT:

- x: usually an element or ideal of self. It should be such that self.ideal(x) is sensible. This excludes x=0.
- degree (default: None): None or an integer. If one, find a prime above x of any degree. If an integer, find a prime above x such that the resulting residue field has exactly this degree.

OUTPUT: A prime ideal of self lying over x. If degree is specified and no such ideal exists, raises a ValueError.

EXAMPLES:

```
sage: x = ZZ['x'].gen()
sage: F.<t> = NumberField(x^3 - 2)

sage: P2 = F.prime_above(2)
sage: P2 # random
Fractional ideal (-t)
sage: 2 in P2
True
sage: P2.is_prime()
True
sage: P2.norm()
2

sage: P3 = F.prime_above(3)
sage: P3 # random
Fractional ideal (t + 1)
sage: 3 in P3
True
sage: P3.is_prime()
True
sage: P3.norm()
3
```

The ideal (3) is totally ramified in F, so there is no degree 2 prime above 3:

```
sage: F.prime_above(3, degree=2)
Traceback (most recent call last):
...
ValueError: No prime of degree 2 above Fractional ideal (3)
sage: [ id.residue_class_degree() for id, _ in F.ideal(3).factor() ]
[1]
```

Asking for a specific degree works:

```
sage: P5_1 = F.prime_above(5, degree=1)
sage: P5_1 # random
Fractional ideal (-t^2 - 1)
sage: P5_1.residue_class_degree()
1

sage: P5_2 = F.prime_above(5, degree=2)
sage: P5_2 # random
Fractional ideal (t^2 - 2*t - 1)
sage: P5_2.residue_class_degree()
```

```
2
```

Relative number fields are ok:

```
sage: G = F.extension(x^2 - 11, 'b')
sage: G.prime_above(7)
Fractional ideal (b + 2)
```

It doesn't make sense to factor the ideal (0):

```
sage: F.prime_above(0)
Traceback (most recent call last):
...
AttributeError: 'NumberFieldIdeal' object has no attribute 'prime_factors'
```

prime_factors(*x*)

Return a list of the prime ideals of self which divide the ideal generated by x.

OUTPUT: list of prime ideals (a new list is returned each time this function is called)

EXAMPLES:

```
sage: K.<w> = NumberField(x^2 + 23)
sage: K.prime_factors(w + 1)
[Fractional ideal (2, 1/2*w - 1/2), Fractional ideal (2, 1/2*w + 1/2), Fractional ideal (3,
```

primes_above(*x*, *degree=None*)

Return prime ideals of self lying over x.

INPUT:

- `x`: usually an element or ideal of self. It should be such that self.ideal(x) is sensible. This excludes x=0.
- `degree` (default: None): None or an integer. If None, find all primes above x of any degree. If an integer, find all primes above x such that the resulting residue field has exactly this degree.

OUTPUT: A list of prime ideals of self lying over x. If degree is specified and no such ideal exists, returns the empty list. The output is sorted by residue degree first, then by underlying prime (or equivalently, by norm).

EXAMPLES:

```
sage: x = ZZ['x'].gen()
sage: F.<t> = NumberField(x^3 - 2)

sage: P2s = F.primes_above(2)
sage: P2s # random
[Fractional ideal (-t)]
sage: all(2 in P2 for P2 in P2s)
True
sage: all(P2.is_prime() for P2 in P2s)
True
sage: [ P2.norm() for P2 in P2s ]
[2]

sage: P3s = F.primes_above(3)
sage: P3s # random
[Fractional ideal (t + 1)]
sage: all(3 in P3 for P3 in P3s)
True
sage: all(P3.is_prime() for P3 in P3s)
```

```
True
sage: [ P3.norm() for P3 in P3s ]
[3]
```

The ideal (3) is totally ramified in F, so there is no degree 2 prime above 3:

```
sage: F.primes_above(3, degree=2)
[]
sage: [ id.residue_class_degree() for id, _ in F.ideal(3).factor() ]
[1]
```

Asking for a specific degree works:

```
sage: P5_1s = F.primes_above(5, degree=1)
sage: P5_1s # random
[Fractional ideal (-t^2 - 1)]
sage: P5_1 = P5_1s[0]; P5_1.residue_class_degree()
1

sage: P5_2s = F.primes_above(5, degree=2)
sage: P5_2s # random
[Fractional ideal (t^2 - 2*t - 1)]
sage: P5_2 = P5_2s[0]; P5_2.residue_class_degree()
2
```

Works in relative extensions too:

```
sage: PQ.<X> = QQ[]
sage: F.<a, b> = NumberField([X^2 - 2, X^2 - 3])
sage: PF.<Y> = F[]
sage: K.<c> = F.extension(Y^2 - (1 + a)*(a + b)*a*b)
sage: I = F.ideal(a + 2*b)
sage: P, Q = K.primes_above(I)
sage: K.ideal(I) == P^4*Q
True
sage: K.primes_above(I, degree=1) == [P]
True
sage: K.primes_above(I, degree=4) == [Q]
True
```

It doesn't make sense to factor the ideal (0), so this raises an error:

```
sage: F.prime_above(0)
Traceback (most recent call last):
...
AttributeError: 'NumberFieldIdeal' object has no attribute 'prime_factors'
```

primes_of_bounded_norm(B)

Returns a sorted list of all prime ideals with norm at most B.

INPUT:

- B – a positive integer; upper bound on the norms of the primes generated.

OUTPUT:

A list of all prime ideals of this number field of norm at most B, sorted by norm. Primes of the same norm are sorted using the comparison function for ideals, which is based on the Hermite Normal Form.

Note: See also primes_of_bounded_norm_iter() for an iterator version of this, but note that the iterator sorts the primes in order of underlying rational prime, not by norm.

EXAMPLES:

```
sage: K.<i> = QuadraticField(-1)
sage: K.primes_of_bounded_norm(10)
[Fractional ideal (i + 1), Fractional ideal (-i - 2), Fractional ideal (2*i + 1), Fractional
sage: K.primes_of_bounded_norm(1)
[]
sage: K.<a> = NumberField(x^3-2)
sage: P = K.primes_of_bounded_norm(30)
sage: P
[Fractional ideal (a),
 Fractional ideal (a + 1),
 Fractional ideal (-a^2 - 1),
 Fractional ideal (a^2 + a - 1),
 Fractional ideal (2*a + 1),
 Fractional ideal (-2*a^2 - a - 1),
 Fractional ideal (a^2 - 2*a - 1),
 Fractional ideal (a + 3)]
sage: [p.norm() for p in P]
[2, 3, 5, 11, 17, 23, 25, 29]
```

primes_of_bounded_norm_iter(*B*)

Iterator yielding all prime ideals with norm at most B.

INPUT:

- B – a positive integer; upper bound on the norms of the primes generated.

OUTPUT:

An iterator over all prime ideals of this number field of norm at most B.

Note: The output is not sorted by norm, but by size of the underlying rational prime.

EXAMPLES:

```
sage: K.<i> = QuadraticField(-1)
sage: it = K.primes_of_bounded_norm_iter(10)
sage: list(it)
[Fractional ideal (i + 1),
 Fractional ideal (3),
 Fractional ideal (-i - 2),
 Fractional ideal (2*i + 1)]
sage: list(K.primes_of_bounded_norm_iter(1))
[]
```

primes_of_degree_one_iter(*num_integer_primes=10000*, *max_iterations=100*)

Return an iterator yielding prime ideals of absolute degree one and small norm.

Warning: It is possible that there are no primes of K of absolute degree one of small prime norm, and it possible that this algorithm will not find any primes of small norm.
See module sage.rings.number_field.small_primes_of_degree_one for details.

INPUT:

- num_integer_primes (default: 10000) - an integer. We try to find primes of absolute norm no greater than the num_integer_primes-th prime number. For example, if num_integer_primes is 2, the largest norm found will be 3, since the second prime is 3.

- max_iterations (default: 100) - an integer. We test max_iterations integers to find small primes before raising StopIteration.

EXAMPLES:

```
sage: K.<z> = CyclotomicField(10)
sage: it = K.primes_of_degree_one_iter()
sage: Ps = [ next(it) for i in range(3) ]
sage: Ps # random
[Fractional ideal (z^3 + z + 1), Fractional ideal (3*z^3 - z^2 + z - 1), Fractional ideal (2
sage: [ P.norm() for P in Ps ] # random
[11, 31, 41]
sage: [ P.residue_class_degree() for P in Ps ]
[1, 1, 1]
```

primes_of_degree_one_list(*n*, *num_integer_primes=10000*, *max_iterations=100*)

Return a list of n prime ideals of absolute degree one and small norm.

Warning: It is possible that there are no primes of K of absolute degree one of small prime norm, and it possible that this algorithm will not find any primes of small norm.
See module sage.rings.number_field.small_primes_of_degree_one for details.

INPUT:

- num_integer_primes (default: 10000) - an integer. We try to find primes of absolute norm no greater than the num_integer_primes-th prime number. For example, if num_integer_primes is 2, the largest norm found will be 3, since the second prime is 3.
- max_iterations (default: 100) - an integer. We test max_iterations integers to find small primes before raising StopIteration.

EXAMPLES:

```
sage: K.<z> = CyclotomicField(10)
sage: Ps = K.primes_of_degree_one_list(3)
sage: Ps  # random output
[Fractional ideal (-z^3 - z^2 + 1), Fractional ideal (2*z^3 - 2*z^2 + 2*z - 3), Fractional i
sage: [ P.norm() for P in Ps ]
[11, 31, 41]
sage: [ P.residue_class_degree() for P in Ps ]
[1, 1, 1]
```

primitive_element()

Return a primitive element for this field, i.e., an element that generates it over $\mathbf{Q}$.

EXAMPLES:

```
sage: K.<a> = NumberField(x^3 + 2)
sage: K.primitive_element()
a
sage: K.<a,b,c> = NumberField([x^2-2,x^2-3,x^2-5])
sage: K.primitive_element()
a - b + c
sage: alpha = K.primitive_element(); alpha
a - b + c
sage: alpha.minpoly()
x^2 + (2*b - 2*c)*x - 2*c*b + 6
sage: alpha.absolute_minpoly()
x^8 - 40*x^6 + 352*x^4 - 960*x^2 + 576
```

primitive_root_of_unity()

Return a generator of the roots of unity in this field.

OUTPUT: a primitive root of unity. No guarantee is made about which primitive root of unity this returns, not even for cyclotomic fields.

Note: We do not create the full unit group since that can be expensive, but we do use it if it is already known.

ALGORITHM:

We use the PARI function `nfrootsof1` in all cases. This is required (even for cyclotomic fields) in order to be consistent with the full unit group, which is also computed by PARI.

EXAMPLES:

```
sage: K.<i> = NumberField(x^2+1)
sage: z = K.primitive_root_of_unity(); z
i
sage: z.multiplicative_order()
4

sage: K.<a> = NumberField(x^2+x+1)
sage: z = K.primitive_root_of_unity(); z
a + 1
sage: z.multiplicative_order()
6

sage: x = polygen(QQ)
sage: F.<a,b> = NumberField([x^2 - 2, x^2 - 3])
sage: y = polygen(F)
sage: K.<c> = F.extension(y^2 - (1 + a)*(a + b)*a*b)
sage: K.primitive_root_of_unity()
-1
```

We do not special-case cyclotomic fields, so we do not always get the most obvious primitive root of unity:

```
sage: K.<a> = CyclotomicField(3)
sage: z = K.primitive_root_of_unity(); z
a + 1
sage: z.multiplicative_order()
6

sage: K = CyclotomicField(3)
sage: z = K.primitive_root_of_unity(); z
zeta3 + 1
sage: z.multiplicative_order()
6
```

TESTS:

Check for trac ticket #15027. We use a new variable name:

```
sage: K.<f> = QuadraticField(-3)
sage: K.primitive_root_of_unity()
-1/2*f + 1/2
sage: UK = K.unit_group()
sage: K.primitive_root_of_unity()
-1/2*f + 1/2
```

random_element(*num_bound=None*, *den_bound=None*, *integral_coefficients=False*, *distribution=None*)

Return a random element of this number field.

INPUT:

- **num_bound - Bound on numerator of the coefficients of** the resulting element
- **den_bound - Bound on denominators of the coefficients** of the resulting element
- **integral_coefficients (default: False) - If True, then** the resulting element will have integral coefficients. This option overrides any value of den_bound.
- **distribution - Distribution to use for the coefficients** of the resulting element

OUTPUT:

- Element of this number field

EXAMPLES:

```
sage: K.<j> = NumberField(x^8+1)
sage: K.random_element()
1/2*j^7 - j^6 - 12*j^5 + 1/2*j^4 - 1/95*j^3 - 1/2*j^2 - 4

sage: K.<a,b,c> = NumberField([x^2-2,x^2-3,x^2-5])
sage: K.random_element()
((6136*c - 7489/3)*b + 5825/3*c - 71422/3)*a + (-4849/3*c + 58918/3)*b - 45718/3*c + 75409/1

sage: K.<a> = NumberField(x^5-2)
sage: K.random_element(integral_coefficients=True)
a^3 + a^2 - 3*a - 1
```

TESTS:

```
sage: K.<a> = NumberField(x^5-2)
sage: K.random_element(-1)
Traceback (most recent call last):
...
TypeError: x must be < y
sage: K.random_element(5,0)
Traceback (most recent call last):
...
TypeError: x must be < y
sage: QQ[I].random_element(0)
Traceback (most recent call last):
...
TypeError: x must be > 0
```

real_embeddings(*prec=53*)

Return all homomorphisms of this number field into the approximate real field with precision prec.

If prec is 53 (the default), then the real double field is used; otherwise the arbitrary precision (but slow) real field is used. If you want embeddings into the 53-bit double precision, which is faster, use `self.embeddings(RDF)`.

Note: This function uses finite precision real numbers. In functions that should output proven results, one could use `self.embeddings(AA)` instead.

EXAMPLES:

```
sage: K.<a> = NumberField(x^3 + 2)
sage: K.real_embeddings()
[
Ring morphism:
  From: Number Field in a with defining polynomial x^3 + 2
  To:   Real Field with 53 bits of precision
  Defn: a |--> -1.25992104989487
]
sage: K.real_embeddings(16)
[
Ring morphism:
  From: Number Field in a with defining polynomial x^3 + 2
  To:   Real Field with 16 bits of precision
  Defn: a |--> -1.260
]
sage: K.real_embeddings(100)
[
Ring morphism:
  From: Number Field in a with defining polynomial x^3 + 2
  To:   Real Field with 100 bits of precision
  Defn: a |--> -1.2599210498948731647672106073
]
```

As this is a numerical function, the number of embeddings may be incorrect if the precision is too low:

```
sage: K = NumberField(x^2+2*10^1000*x + 10^2000+1, 'a')
sage: len(K.real_embeddings())
2
sage: len(K.real_embeddings(100))
2
sage: len(K.real_embeddings(10000))
0
sage: len(K.embeddings(AA))
0
```

reduced_basis(*prec=None*)

This function returns an LLL-reduced basis for the Minkowski-embedding of the maximal order of a number field.

INPUT:

- `self` - number field, the base field
- `prec (default: None)` - the precision with which to compute the Minkowski embedding.

OUTPUT:

An LLL-reduced basis for the Minkowski-embedding of the maximal order of a number field, given by a sequence of (integral) elements from the field.

Note: In the non-totally-real case, the LLL routine we call is currently PARI's qflll(), which works with floating point approximations, and so the result is only as good as the precision promised by PARI. The matrix returned will always be integral; however, it may only be only "almost" LLL-reduced when the precision is not sufficiently high.

EXAMPLES:

```
sage: F.<t> = NumberField(x^6-7*x^4-x^3+11*x^2+x-1)
sage: F.maximal_order().basis()
[1/2*t^5 + 1/2*t^4 + 1/2*t^2 + 1/2, t, t^2, t^3, t^4, t^5]
```

```
sage: F.reduced_basis()
[-1, -1/2*t^5 + 1/2*t^4 + 3*t^3 - 3/2*t^2 - 4*t - 1/2, t, 1/2*t^5 + 1/2*t^4 - 4*t^3 - 5/2*t^
sage: CyclotomicField(12).reduced_basis()
[1, zeta12^2, zeta12, zeta12^3]
```

reduced_gram_matrix(*prec=None*)

This function returns the Gram matrix of an LLL-reduced basis for the Minkowski embedding of the maximal order of a number field.

INPUT:

- `self` - number field, the base field
- `prec (default: None)` - the precision with which to calculate the Minkowski embedding. (See NOTE below.)

OUTPUT: The Gram matrix $[\langle x_i, x_j \rangle]$ of an LLL reduced basis for the maximal order of self, where the integral basis for self is given by $\{x_0, \ldots, x_{n-1}\}$. Here $\langle , \rangle$ is the usual inner product on $\mathbf{R}^n$, and self is embedded in $\mathbf{R}^n$ by the Minkowski embedding. See the docstring for `NumberField_absolute.Minkowski_embedding()` for more information.

Note: In the non-totally-real case, the LLL routine we call is currently PARI's qflll(), which works with floating point approximations, and so the result is only as good as the precision promised by PARI. In particular, in this case, the returned matrix will *not* be integral, and may not have enough precision to recover the correct gram matrix (which is known to be integral for theoretical reasons). Thus the need for the prec flag above.

If the following run-time error occurs: "PariError: not a definite matrix in lllgram (42)" try increasing the prec parameter,

EXAMPLES:

```
sage: F.<t> = NumberField(x^6-7*x^4-x^3+11*x^2+x-1)
sage: F.reduced_gram_matrix()
[ 6  3  0  2  0  1]
[ 3  9  0  1  0 -2]
[ 0  0 14  6 -2  3]
[ 2  1  6 16 -3  3]
[ 0  0 -2 -3 16  6]
[ 1 -2  3  3  6 19]
sage: Matrix(6, [(x*y).trace() for x in F.integral_basis() for y in F.integral_basis()])
[2550  133  259  664 1368 3421]
[ 133   14    3   54   30  233]
[ 259    3   54   30  233  217]
[ 664   54   30  233  217 1078]
[1368   30  233  217 1078 1371]
[3421  233  217 1078 1371 5224]

sage: var('x')
x
sage: F.<alpha> = NumberField(x^4+x^2+712312*x+131001238)
sage: F.reduced_gram_matrix(prec=128)
[   4.000000000000000000000000000000000000   0.000000000000000000000000000000000000 -2.13
[  0.000000000000000000000000000000000000    46721.539331563218381658483353092335550 -2.24
[-2.136936000000000000000000000000000000e6 -2.246776905739453010909475522339581932e7 7.070
[-3.312247800000000000000000000000000000e7 -3.480727604113845047361162908864749643e8 1.125
```

regulator(*proof=None*)

Return the regulator of this number field.

Note that PARI computes the regulator to higher precision than the Sage default.

INPUT:

- `proof` - default: True, unless you set it otherwise.

EXAMPLES:

```
sage: NumberField(x^2-2, 'a').regulator()
0.881373587019543
sage: NumberField(x^4+x^3+x^2+x+1, 'a').regulator()
0.962423650119207
```

residue_field(*prime*, *names=None*, *check=True*)

Return the residue field of this number field at a given prime, ie O_K/pO_K.

INPUT:

- `prime` - a prime ideal of the maximal order in this number field, or an element of the field which generates a principal prime ideal.
- `names` - the name of the variable in the residue field
- `check` - whether or not to check the primality of prime.

OUTPUT: The residue field at this prime.

EXAMPLES:

```
sage: R.<x> = QQ[]
sage: K.<a> = NumberField(x^4+3*x^2-17)
sage: P = K.ideal(61).factor()[0][0]
sage: K.residue_field(P)
Residue field in abar of Fractional ideal (61, a^2 + 30)

sage: K.<i> = NumberField(x^2 + 1)
sage: K.residue_field(1+i)
Residue field of Fractional ideal (i + 1)
```

TESTS:

```
sage: L.<b> = NumberField(x^2 + 5)
sage: L.residue_field(P)
Traceback (most recent call last):
...
ValueError: Fractional ideal (61, a^2 + 30) is not an ideal of Number Field in b with defini
sage: L.residue_field(2)
Traceback (most recent call last):
...
ValueError: Fractional ideal (2) is not a prime ideal

sage: L.residue_field(L.prime_above(5)^2)
Traceback (most recent call last):
...
ValueError: Fractional ideal (5) is not a prime ideal
```

roots_of_unity()

Return all the roots of unity in this field, primitive or not.

EXAMPLES:

```
sage: K.<b> = NumberField(x^2+1)
sage: zs = K.roots_of_unity(); zs
[b, -1, -b, 1]
```

```
sage: [ z**K.number_of_roots_of_unity() for z in zs ]
[1, 1, 1, 1]
```

selmer_group(*S*, *m*, *proof=True*, *orders=False*)

Compute the group $K(S, m)$.

INPUT:

- `S` – a set of primes of `self`
- `m` – a positive integer
- `proof` – if False, assume the GRH in computing the class group
- `orders` (default False) – if True, output two lists, the generators and their orders

OUTPUT:

A list of generators of $K(S, m)$, and (optionally) their orders as elements of $K^\times/(K^\times)^m$. This is the subgroup of $K^\times/(K^\times)^m$ consisting of elements a such that the valuation of a is divisible by m at all primes not in S. It fits in an exact sequence between the units modulo m-th powers and the m-torsion in the S-class group:

$$1 \longrightarrow O_{K,S}^\times/(O_{K,S}^\times)^m \longrightarrow K(S,m) \longrightarrow \mathrm{Cl}_{K,S}[m] \longrightarrow 0.$$

The group $K(S, m)$ contains the subgroup of those a such that $K(\sqrt[m]{a})/K$ is unramified at all primes of K outside of S, but may contain it properly when not all primes dividing m are in S.

EXAMPLES:

```
sage: K.<a> = QuadraticField(-5)
sage: K.selmer_group((), 2)
[-1, 2]
```

The previous example shows that the group generated by the output may be strictly larger than the 'true' Selmer group of elements giving extensions unramified outside S, since that has order just 2, generated by -1:

```
sage: K.class_number()
2
sage: K.hilbert_class_field('b')
Number Field in b with defining polynomial x^2 + 1 over its base field
```

When m is prime all the orders are equal to m, but in general they are only divisors of m:

```
sage: K.<a> = QuadraticField(-5)
sage: P2 = K.ideal(2, -a+1)
sage: P3 = K.ideal(3, a+1)
sage: K.selmer_group((), 2, orders=True)
([-1, 2], [2, 2])
sage: K.selmer_group((), 4, orders=True)
([-1, 4], [2, 2])
sage: K.selmer_group([P2], 2)
[2, -1]
sage: K.selmer_group((P2,P3), 4)
[2, a + 1, -1]
sage: K.selmer_group((P2,P3), 4, orders=True)
([2, a + 1, -1], [4, 4, 2])
sage: K.selmer_group([P2], 3)
[2]
sage: K.selmer_group([P2, P3], 3)
[2, a + 1]
```

```
sage: K.selmer_group([P2, P3, K.ideal(a)], 3)  # random signs
[2, a + 1, a]
```

Example over $\mathbf{Q}$ (as a number field):

```
sage: K.<a> = NumberField(polygen(QQ))
sage: K.selmer_group([],5)
[]
sage: K.selmer_group([K.prime_above(p) for p in [2,3,5]],2)
[2, 3, 5, -1]
sage: K.selmer_group([K.prime_above(p) for p in [2,3,5]],6, orders=True)
([2, 3, 5, -1], [6, 6, 6, 2])
```

TESTS:

```
sage: K.<a> = QuadraticField(-5)
sage: P2 = K.ideal(2, -a+1)
sage: P3 = K.ideal(3, a+1)
sage: P5 = K.ideal(a)
sage: S = K.selmer_group([P2, P3, P5], 3)
sage: S == [2, a + 1, a] or S == [2, a + 1, -a]
True
```

Verify that trac ticket #14489 is fixed:

```
sage: K.<a> = NumberField(x^3 - 381 * x + 127)
sage: K.selmer_group(K.primes_above(13), 2)
[-7/13*a^2 - 140/13*a + 36/13, 14/13*a^2 + 267/13*a - 85/13, 7/13*a^2 + 127/13*a - 49/13, -1
```

Verify that trac ticket #16708 is fixed:

```
sage: K.<a> = QuadraticField(-5)
sage: p = K.primes_above(2)[0]
sage: S = K.selmer_group((), 4)
sage: all(4.divides(x.valuation(p)) for x in S)
True
```

selmer_group_iterator(*S*, *m*, *proof=True*)

Return an iterator through elements of the finite group $K(S, m)$.

INPUT:

- `S` – a set of primes of `self`
- `m` – a positive integer
- `proof` – if False, assume the GRH in computing the class group

OUTPUT:

An iterator yielding the distinct elements of $K(S, m)$. See the docstring for `NumberField_generic.selmer_group()` for more information.

EXAMPLES:

```
sage: K.<a> = QuadraticField(-5)
sage: list(K.selmer_group_iterator((), 2))
[1, 2, -1, -2]
sage: list(K.selmer_group_iterator((), 4))
[1, 4, -1, -4]
sage: list(K.selmer_group_iterator([K.ideal(2, -a+1)], 2))
[1, -1, 2, -2]
```

```
sage: list(K.selmer_group_iterator([K.ideal(2, -a+1), K.ideal(3, a+1)], 2))
[1, -1, a + 1, -a - 1, 2, -2, 2*a + 2, -2*a - 2]
```

Examples over $\mathbf{Q}$ (as a number field):

```
sage: K.<a> = NumberField(polygen(QQ))
sage: list(K.selmer_group_iterator([], 5))
[1]
sage: list(K.selmer_group_iterator([], 4))
[1, -1]
sage: list(K.selmer_group_iterator([K.prime_above(p) for p in [11,13]],2))
[1, -1, 13, -13, 11, -11, 143, -143]
```

signature()

Return (r1, r2), where r1 and r2 are the number of real embeddings and pairs of complex embeddings of this field, respectively.

EXAMPLES:

```
sage: NumberField(x^2+1, 'a').signature()
(0, 1)
sage: NumberField(x^3-2, 'a').signature()
(1, 1)
```

solve_CRT(*reslist*, *Ilist*, *check=True*)

Solve a Chinese remainder problem over this number field.

INPUT:

- `reslist` – a list of residues, i.e. integral number field elements
- `Ilist` – a list of integral ideals, assumed pairsise coprime
- `check` (boolean, default True) – if True, result is checked

OUTPUT:

An integral element x such that x-reslist[i] is in Ilist[i] for all i.

Note: The current implementation requires the ideals to be pairwise coprime. A more general version would be possible.

EXAMPLES:

```
sage: K.<a> = NumberField(x^2-10)
sage: Ilist = [K.primes_above(p)[0] for p in prime_range(10)]
sage: b = K.solve_CRT([1,2,3,4],Ilist,True)
sage: all([b-i-1 in Ilist[i] for i in range(4)])
True
sage: Ilist = [K.ideal(a), K.ideal(2)]
sage: K.solve_CRT([0,1],Ilist,True)
Traceback (most recent call last):
...
ArithmeticError: ideals in solve_CRT() must be pairwise coprime
sage: Ilist[0]+Ilist[1]
Fractional ideal (2, a)
```

specified_complex_embedding()

Returns the embedding of this field into the complex numbers which has been specified.

Fields created with the `QuadraticField` or `CyclotomicField` constructors come with an implicit embedding. To get one of these fields without the embedding, use the generic `NumberField` constructor.

EXAMPLES:

```
sage: QuadraticField(-1, 'I').specified_complex_embedding()
Generic morphism:
  From: Number Field in I with defining polynomial x^2 + 1
  To:   Complex Lazy Field
  Defn: I -> 1*I

sage: QuadraticField(3, 'a').specified_complex_embedding()
Generic morphism:
  From: Number Field in a with defining polynomial x^2 - 3
  To:   Real Lazy Field
  Defn: a -> 1.732050807568878?

sage: CyclotomicField(13).specified_complex_embedding()
Generic morphism:
  From: Cyclotomic Field of order 13 and degree 12
  To:   Complex Lazy Field
  Defn: zeta13 -> 0.885456025653210? + 0.464723172043769?*I
```

Most fields don't implicitly have embeddings unless explicitly specified:

```
sage: NumberField(x^2-2, 'a').specified_complex_embedding() is None
True
sage: NumberField(x^3-x+5, 'a').specified_complex_embedding() is None
True
sage: NumberField(x^3-x+5, 'a', embedding=2).specified_complex_embedding()
Generic morphism:
  From: Number Field in a with defining polynomial x^3 - x + 5
  To:   Real Lazy Field
  Defn: a -> -1.904160859134921?
sage: NumberField(x^3-x+5, 'a', embedding=CDF.0).specified_complex_embedding()
Generic morphism:
  From: Number Field in a with defining polynomial x^3 - x + 5
  To:   Complex Lazy Field
  Defn: a -> 0.952080429567461? + 1.311248044077123?*I
```

This function only returns complex embeddings:

```
sage: K.<a> = NumberField(x^2-2, embedding=Qp(7)(2).sqrt())
sage: K.specified_complex_embedding() is None
True
sage: K.gen_embedding()
3 + 7 + 2*7^2 + 6*7^3 + 7^4 + 2*7^5 + 7^6 + 2*7^7 + 4*7^8 + 6*7^9 + 6*7^10 + 2*7^11 + 7^12 +
sage: K.coerce_embedding()
Generic morphism:
  From: Number Field in a with defining polynomial x^2 - 2
  To:   7-adic Field with capped relative precision 20
  Defn: a -> 3 + 7 + 2*7^2 + 6*7^3 + 7^4 + 2*7^5 + 7^6 + 2*7^7 + 4*7^8 + 6*7^9 + 6*7^10 + 2*
```

structure()

Return fixed isomorphism or embedding structure on self.

This is used to record various isomorphisms or embeddings that arise naturally in other constructions.

EXAMPLES:

```
sage: K.<z> = NumberField(x^2 + 3)
sage: L.<a> = K.absolute_field(); L
Number Field in a with defining polynomial x^2 + 3
sage: L.structure()
(Isomorphism given by variable name change map:
  From: Number Field in a with defining polynomial x^2 + 3
  To:   Number Field in z with defining polynomial x^2 + 3,
 Isomorphism given by variable name change map:
  From: Number Field in z with defining polynomial x^2 + 3
  To:   Number Field in a with defining polynomial x^2 + 3)

sage: K.<a> = QuadraticField(-3)
sage: R.<y> = K[]
sage: D.<x0> = K.extension(y)
sage: D_abs.<y0> = D.absolute_field()
sage: D_abs.structure()[0](y0)
-a
```

subfield(*alpha*, *name=None*, *names=None*)

Return a number field K isomorphic to $\mathbf{Q}(\alpha)$ (if this is an absolute number field) or $L(\alpha)$ (if this is a relative extension M/L) and a map from K to self that sends the generator of K to alpha.

INPUT:

- `alpha` - an element of self, or something that coerces to an element of self.

OUTPUT:

- `K` - a number field
- `from_K` - a homomorphism from K to self that sends the generator of K to alpha.

EXAMPLES:

```
sage: K.<a> = NumberField(x^4 - 3); K
Number Field in a with defining polynomial x^4 - 3
sage: H.<b>, from_H = K.subfield(a^2)
sage: H
Number Field in b with defining polynomial x^2 - 3
sage: from_H(b)
a^2
sage: from_H
Ring morphism:
  From: Number Field in b with defining polynomial x^2 - 3
  To:   Number Field in a with defining polynomial x^4 - 3
  Defn: b |--> a^2
```

A relative example. Note that the result returned is the subfield generated by α over `self.base_field()`, not over $\mathbf{Q}$ (see trac ticket #5392):

```
sage: L.<a> = NumberField(x^2 - 3)
sage: M.<b> = L.extension(x^4 + 1)
sage: K, phi = M.subfield(b^2)
sage: K.base_field() is L
True
```

Subfields inherit embeddings:

```
sage: K.<z> = CyclotomicField(5)
sage: L, K_from_L = K.subfield(z-z^2-z^3+z^4)
sage: L
```

```
Number Field in z0 with defining polynomial x^2 - 5
sage: CLF_from_K = K.coerce_embedding(); CLF_from_K
Generic morphism:
  From: Cyclotomic Field of order 5 and degree 4
  To:   Complex Lazy Field
  Defn: z -> 0.309016994374948? + 0.951056516295154?*I
sage: CLF_from_L = L.coerce_embedding(); CLF_from_L
Generic morphism:
  From: Number Field in z0 with defining polynomial x^2 - 5
  To:   Complex Lazy Field
  Defn: z0 -> 2.23606797749979? + 0.?e-14*I
```

Check transitivity:

```
sage: CLF_from_L(L.gen())
2.23606797749979? + 0.?e-14*I
sage: CLF_from_K(K_from_L(L.gen()))
2.23606797749979? + 0.?e-14*I
```

If $self$ has no specified embedding, then K comes with an embedding in $self$:

```
sage: K.<a> = NumberField(x^6 - 6*x^4 + 8*x^2 - 1)
sage: L.<b>, from_L = K.subfield(a^2)
sage: L
Number Field in b with defining polynomial x^3 - 6*x^2 + 8*x - 1
sage: L.gen_embedding()
a^2
```

You can also view a number field as having a different generator by just choosing the input to generate the whole field; for that it is better to use `self.change_generator`, which gives isomorphisms in both directions.

trace_dual_basis(b)

Compute the dual basis of a basis of `self` with respect to the trace pairing.

EXAMPLES:

```
sage: K.<a> = NumberField(x^3 + x + 1)
sage: b = [1, 2*a, 3*a^2]
sage: T = K.trace_dual_basis(b); T
[4/31*a^2 - 6/31*a + 13/31, -9/62*a^2 - 1/31*a - 3/31, 2/31*a^2 - 3/31*a + 4/93]
sage: [(b[i]*T[j]).trace() for i in xrange(3) for j in xrange(3)]
[1, 0, 0, 0, 1, 0, 0, 0, 1]
```

trace_pairing(v)

Return the matrix of the trace pairing on the elements of the list v.

EXAMPLES:

```
sage: K.<zeta3> = NumberField(x^2 + 3)
sage: K.trace_pairing([1,zeta3])
[ 2  0]
[ 0 -6]
```

uniformizer(P, *others='positive'*)

Returns an element of self with valuation 1 at the prime ideal P.

INPUT:

- `self` - a number field

- P - a prime ideal of self
- others - either "positive" (default), in which case the element will have non-negative valuation at all other primes of self, or "negative", in which case the element will have non-positive valuation at all other primes of self.

Note: When P is principal (e.g. always when self has class number one) the result may or may not be a generator of P!

EXAMPLES:

```
sage: K.<a> = NumberField(x^2 + 5); K
Number Field in a with defining polynomial x^2 + 5
sage: P,Q = K.ideal(3).prime_factors()
sage: P
Fractional ideal (3, a + 1)
sage: pi = K.uniformizer(P); pi
a + 1
sage: K.ideal(pi).factor()
(Fractional ideal (2, a + 1)) * (Fractional ideal (3, a + 1))
sage: pi = K.uniformizer(P,'negative'); pi
1/2*a + 1/2
sage: K.ideal(pi).factor()
(Fractional ideal (2, a + 1))^-1 * (Fractional ideal (3, a + 1))

sage: K = CyclotomicField(9)
sage: Plist=K.ideal(17).prime_factors()
sage: pilist = [K.uniformizer(P) for P in Plist]
sage: [pi.is_integral() for pi in pilist]
[True, True, True]
sage: [pi.valuation(P) for pi,P in zip(pilist,Plist)]
[1, 1, 1]
sage: [ pilist[i] in Plist[i] for i in range(len(Plist)) ]
[True, True, True]

sage: K.<t> = NumberField(x^4 - x^3 - 3*x^2 - x + 1)
sage: [K.uniformizer(P) for P,e in factor(K.ideal(2))]
[2]
sage: [K.uniformizer(P) for P,e in factor(K.ideal(3))]
[t - 1]
sage: [K.uniformizer(P) for P,e in factor(K.ideal(5))]
[t^2 - t + 1, t + 2, t - 2]
sage: [K.uniformizer(P) for P,e in factor(K.ideal(7))]
[t^2 + 3*t + 1]
sage: [K.uniformizer(P) for P,e in factor(K.ideal(67))]
[t + 23, t + 26, t - 32, t - 18]
```

ALGORITHM:

Use PARI. More precisely, use the second component of idealprimedec in the "positive" case. Use *idealappr* with exponent of -1 and invert the result in the "negative" case.

unit_group(*proof=None*)

Return the unit group (including torsion) of this number field.

ALGORITHM: Uses PARI's bnfunit command.

INPUT:

- proof (bool, default True) flag passed to pari.

Note: The group is cached.

EXAMPLES:

```
sage: x = QQ['x'].0
sage: A = x^4 - 10*x^3 + 20*5*x^2 - 15*5^2*x + 11*5^3
sage: K = NumberField(A, 'a')
sage: U = K.unit_group(); U
Unit group with structure C10 x Z of Number Field in a with defining polynomial x^4 - 10*x^3
sage: U.gens()
(u0, u1)
sage: U.gens_values()
[-7/275*a^3 + 1/11*a^2 - 9/11*a - 1, 7/275*a^3 - 1/11*a^2 + 9/11*a + 2]
sage: U.invariants()
(10, 0)
sage: [u.multiplicative_order() for u in U.gens()]
[10, +Infinity]
```

For big number fields, provably computing the unit group can take a very long time. In this case, one can ask for the conjectural unit group (correct if the Generalized Riemann Hypothesis is true):

```
sage: K = NumberField(x^17 + 3, 'a')
sage: K.unit_group(proof=True)  # takes forever, not tested
...
sage: U = K.unit_group(proof=False)
sage: U
Unit group with structure C2 x Z x Z x Z x Z x Z x Z x Z x Z of Number Field in a with defin
sage: U.gens()
(u0, u1, u2, u3, u4, u5, u6, u7, u8)
sage: U.gens_values()  # result not independently verified
[-1, a^9 + a - 1, a^15 - a^12 + a^10 - a^9 - 2*a^8 + 3*a^7 + a^6 - 3*a^5 + a^4 + 4*a^3 - 3*a
```

units (*proof=None*)

Return generators for the unit group modulo torsion.

INPUT:

- `proof` (bool, default True) flag passed to `pari`.

Note: For more functionality see the unit_group() function.

ALGORITHM: Uses PARI's bnfunit command.

EXAMPLES:

```
sage: x = polygen(QQ)
sage: A = x^4 - 10*x^3 + 20*5*x^2 - 15*5^2*x + 11*5^3
sage: K = NumberField(A, 'a')
sage: K.units()
(7/275*a^3 - 1/11*a^2 + 9/11*a + 2,)
```

For big number fields, provably computing the unit group can take a very long time. In this case, one can ask for the conjectural unit group (correct if the Generalized Riemann Hypothesis is true):

```
sage: K = NumberField(x^17 + 3, 'a')
sage: K.units(proof=True)  # takes forever, not tested
...
sage: K.units(proof=False)  # result not independently verified
(a^9 + a - 1,
```

```
a^15 - a^12 + a^10 - a^9 - 2*a^8 + 3*a^7 + a^6 - 3*a^5 + a^4 + 4*a^3 - 3*a^2 - 2*a + 2,
a^16 - a^15 + a^14 - a^12 + a^11 - a^10 - a^8 + a^7 - 2*a^6 + a^4 - 3*a^3 + 2*a^2 - 2*a + 1
2*a^16 - a^14 - a^13 + 3*a^12 - 2*a^10 + a^9 + 3*a^8 - 3*a^6 + 3*a^5 + 3*a^4 - 2*a^3 - 2*a^
2*a^16 - 3*a^15 + 3*a^14 - 3*a^13 + 3*a^12 - a^11 + a^9 - 3*a^8 + 4*a^7 - 5*a^6 + 6*a^5 - 4
a^16 - a^15 - 3*a^14 - 4*a^13 - 4*a^12 - 3*a^11 - a^10 + 2*a^9 + 4*a^8 + 5*a^7 + 4*a^6 + 2*
a^15 + a^14 + 2*a^11 + a^10 - a^9 + a^8 + 2*a^7 - a^5 + 2*a^3 - a^2 - 3*a + 1,
5*a^16 - 6*a^14 + a^13 + 7*a^12 - 2*a^11 - 7*a^10 + 4*a^9 + 7*a^8 - 6*a^7 - 7*a^6 + 8*a^5 +
```

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

Return one, or a list of all, primitive n-th root of unity in this field.

INPUT:

- `n` - positive integer
- `all` - bool. If False (default), return a primitive n-th root of unity in this field, or raise a ValueError exception if there are none. If True, return a list of all primitive n-th roots of unity in this field (possibly empty).

Note: To obtain the maximal order of a root of unity in this field, use self.number_of_roots_of_unity().

Note: We do not create the full unit group since that can be expensive, but we do use it if it is already known.

EXAMPLES:

```
sage: K.<z> = NumberField(x^2 + 3)
sage: K.zeta(1)
1
sage: K.zeta(2)
-1
sage: K.zeta(2, all=True)
[-1]
sage: K.zeta(3)
1/2*z - 1/2
sage: K.zeta(3, all=True)
[1/2*z - 1/2, -1/2*z - 1/2]
sage: K.zeta(4)
Traceback (most recent call last):
...
ValueError: There are no 4th roots of unity in self.

sage: r.<x> = QQ[]
sage: K.<b> = NumberField(x^2+1)
sage: K.zeta(4)
b
sage: K.zeta(4,all=True)
[b, -b]
sage: K.zeta(3)
Traceback (most recent call last):
...
ValueError: There are no 3rd roots of unity in self.
sage: K.zeta(3,all=True)
[]
```

zeta_coefficients (*n*)

Compute the first n coefficients of the Dedekind zeta function of this field as a Dirichlet series.

EXAMPLE:

```
sage: x = QQ['x'].0
sage: NumberField(x^2+1, 'a').zeta_coefficients(10)
[1, 1, 0, 1, 2, 0, 0, 1, 1, 2]
```

zeta_function(*prec=53*, *max_imaginary_part=0*, *max_asymp_coeffs=40*)

Return the Zeta function of this number field.

This actually returns an interface to Tim Dokchitser's program for computing with the Dedekind zeta function zeta_F(s) of the number field F.

INPUT:

- `prec` - integer (bits precision)
- `max_imaginary_part` - real number
- `max_asymp_coeffs` - integer

OUTPUT: The zeta function of this number field.

EXAMPLES:

```
sage: K.<a> = NumberField(ZZ['x'].0^2+ZZ['x'].0-1)
sage: Z = K.zeta_function()
sage: Z
Zeta function associated to Number Field in a with defining polynomial x^2 + x - 1
sage: Z(-1)
0.0333333333333333
sage: L.<a, b, c> = NumberField([x^2 - 5, x^2 + 3, x^2 + 1])
sage: Z = L.zeta_function()
sage: Z(5)
1.00199015670185
```

zeta_order()

Return the number of roots of unity in this field.

Note: We do not create the full unit group since that can be expensive, but we do use it if it is already known.

EXAMPLES:

```
sage: F.<alpha> = NumberField(x**22+3)
sage: F.zeta_order()
6
sage: F.<alpha> = NumberField(x**2-7)
sage: F.zeta_order()
2
```

sage.rings.number_field.number_field.**NumberField_generic_v1**(*poly*, *name*, *latex_name*, *canonical_embedding=None*)

This is used in pickling generic number fields.

EXAMPLES:

```
sage: from sage.rings.number_field.number_field import NumberField_absolute_v1
sage: R.<x> = QQ[]
sage: NumberField_absolute_v1(x^2 + 1, 'i', 'i')
Number Field in i with defining polynomial x^2 + 1
```

class sage.rings.number_field.number_field.**NumberField_quadratic**(*polynomial*, *name=None*, *latex_name=None*, *check=True*, *embedding=None*, *assume_disc_small=False*, *maximize_at_primes=None*, *structure=None*)

Bases: sage.rings.number_field.number_field.NumberField_absolute

Create a quadratic extension of the rational field.

The command `QuadraticField(a)` creates the field $\mathbf{Q}(\sqrt{a})$.

EXAMPLES:

```
sage: QuadraticField(3, 'a')
Number Field in a with defining polynomial x^2 - 3
sage: QuadraticField(-4, 'b')
Number Field in b with defining polynomial x^2 + 4
```

class_number(*proof=None*)

Return the size of the class group of self.

If proof = False (*not* the default!) and the discriminant of the field is negative, then the following warning from the PARI manual applies:

> **Warning:** For $D < 0$, this function may give incorrect results when the class group has a low exponent (has many cyclic factors), because implementing Shank's method in full generality slows it down immensely.

EXAMPLES:

```
sage: QuadraticField(-23,'a').class_number()
3
```

These are all the primes so that the class number of $\mathbf{Q}(\sqrt{-p})$ is 1:

```
sage: [d for d in prime_range(2,300) if not is_square(d) and QuadraticField(-d,'a').class_nu
[2, 3, 7, 11, 19, 43, 67, 163]
```

It is an open problem to *prove* that there are infinity many positive square-free d such that $\mathbf{Q}(\sqrt{d})$ has class number 1:

```
sage: len([d for d in range(2,200) if not is_square(d) and QuadraticField(d,'a').class_numbe
121
```

TESTS:

```
sage: type(QuadraticField(-23,'a').class_number())
<type 'sage.rings.integer.Integer'>
sage: type(NumberField(x^3 + 23, 'a').class_number())
<type 'sage.rings.integer.Integer'>
sage: type(NumberField(x^3 + 23, 'a').extension(x^2 + 5, 'b').class_number())
<type 'sage.rings.integer.Integer'>
sage: type(CyclotomicField(10).class_number())
<type 'sage.rings.integer.Integer'>
```

discriminant(*v=None*)

Returns the discriminant of the ring of integers of the number field, or if v is specified, the determinant of the trace pairing on the elements of the list v.

INPUT:

- v (optional) - list of element of this number field

OUTPUT: Integer if v is omitted, and Rational otherwise.

EXAMPLES:

```
sage: K.<i> = NumberField(x^2+1)
sage: K.discriminant()
-4
sage: K.<a> = NumberField(x^2+5)
sage: K.discriminant()
-20
sage: K.<a> = NumberField(x^2-5)
sage: K.discriminant()
5
```

hilbert_class_field(*names*)

Returns the Hilbert class field of this quadratic field as a relative extension of this field.

Note: For the polynomial that defines this field as a relative extension, see the hilbert_class_field_defining_polynomial command, which is vastly faster than this command, since it doesn't construct a relative extension.

EXAMPLES:

```
sage: K.<a> = NumberField(x^2 + 23)
sage: L = K.hilbert_class_field('b'); L
Number Field in b with defining polynomial x^3 - x^2 + 1 over its base field
sage: L.absolute_field('c')
Number Field in c with defining polynomial x^6 - 2*x^5 + 70*x^4 - 90*x^3 + 1631*x^2 - 1196*x
sage: K.hilbert_class_field_defining_polynomial()
x^3 - x^2 + 1
```

hilbert_class_field_defining_polynomial(*name='x'*)

Returns a polynomial over $\mathbf{Q}$ whose roots generate the Hilbert class field of this quadratic field as an extension of this quadratic field.

Note: Computed using PARI via Schertz's method. This implementation is quite fast.

EXAMPLES:

```
sage: K.<b> = QuadraticField(-23)
sage: K.hilbert_class_field_defining_polynomial()
x^3 - x^2 + 1
```

Note that this polynomial is not the actual Hilbert class polynomial: see hilbert_class_polynomial:

```
sage: K.hilbert_class_polynomial()
x^3 + 3491750*x^2 - 5151296875*x + 12771880859375

sage: K.<a> = QuadraticField(-431)
sage: K.class_number()
21
```

```
sage: K.hilbert_class_field_defining_polynomial(name='z')
z^21 + 6*z^20 + 9*z^19 - 4*z^18 + 33*z^17 + 140*z^16 + 220*z^15 + 243*z^14 + 297*z^13 + 461*
```

hilbert_class_polynomial(*name='x'*)

Compute the Hilbert class polynomial of this quadratic field.

Right now, this is only implemented for imaginary quadratic fields.

EXAMPLES:

```
sage: K.<a> = QuadraticField(-3)
sage: K.hilbert_class_polynomial()
x

sage: K.<a> = QuadraticField(-31)
sage: K.hilbert_class_polynomial(name='z')
z^3 + 39491307*z^2 - 58682638134*z + 1566028350940383
```

is_galois()

Return True since all quadratic fields are automatically Galois.

EXAMPLES:

```
sage: QuadraticField(1234,'d').is_galois()
True
```

sage.rings.number_field.number_field.**NumberField_quadratic_v1**(*poly*, *name*, *canonical_embedding=None*)

This is used in pickling quadratic fields.

EXAMPLES:

```
sage: from sage.rings.number_field.number_field import NumberField_quadratic_v1
sage: R.<x> = QQ[]
sage: NumberField_quadratic_v1(x^2 - 2, 'd')
Number Field in d with defining polynomial x^2 - 2
```

sage.rings.number_field.number_field.**QuadraticField**(*D*, *name='a'*, *check=True*, *embedding=True*, *latex_name='sqrt'*, ***args*)

Return a quadratic field obtained by adjoining a square root of D to the rational numbers, where D is not a perfect square.

INPUT:

- `D` - a rational number
- `name` - variable name (default: 'a')
- `check` - bool (default: True)
- `embedding` - bool or square root of D in an ambient field (default: True)
- `latex_name` - latex variable name (default: sqrt{D})

OUTPUT: A number field defined by a quadratic polynomial. Unless otherwise specified, it has an embedding into $\mathbf{R}$ or $\mathbf{C}$ by sending the generator to the positive or upper-half-plane root.

EXAMPLES:

```
sage: QuadraticField(3, 'a')
Number Field in a with defining polynomial x^2 - 3
sage: K.<theta> = QuadraticField(3); K
```

```
Number Field in theta with defining polynomial x^2 - 3
sage: RR(theta)
1.73205080756888
sage: QuadraticField(9, 'a')
Traceback (most recent call last):
...
ValueError: D must not be a perfect square.
sage: QuadraticField(9, 'a', check=False)
Number Field in a with defining polynomial x^2 - 9
```

Quadratic number fields derive from general number fields.

```
sage: from sage.rings.number_field.number_field import is_NumberField
sage: type(K)
<class 'sage.rings.number_field.number_field.NumberField_quadratic_with_category'>
sage: is_NumberField(K)
True
```

Quadratic number fields are cached:

```
sage: QuadraticField(-11, 'a') is QuadraticField(-11, 'a')
True
```

By default, quadratic fields come with a nice latex representation:

```
sage: K.<a> = QuadraticField(-7)
sage: latex(K)
\Bold{Q}(\sqrt{-7})
sage: latex(a)
\sqrt{-7}
sage: latex(1/(1+a))
-\frac{1}{8} \sqrt{-7} + \frac{1}{8}
sage: K.latex_variable_name()
'\\sqrt{-7}'
```

We can provide our own name as well:

```
sage: K.<a> = QuadraticField(next_prime(10^10), latex_name=r'\sqrt{D}')
sage: 1+a
a + 1
sage: latex(1+a)
\sqrt{D} + 1
sage: latex(QuadraticField(-1, 'a', latex_name=None).gen())
a
```

The name of the generator does not interfere with Sage preparser, see #1135:

```
sage: K1 = QuadraticField(5, 'x')
sage: K2.<x> = QuadraticField(5)
sage: K3.<x> = QuadraticField(5, 'x')
sage: K1 is K2
True
sage: K1 is K3
True
sage: K1
Number Field in x with defining polynomial x^2 - 5
```

Note that, in presence of two different names for the generator, the name given by the preparser takes precedence:

```
sage: K4.<y> = QuadraticField(5, 'x'); K4
Number Field in y with defining polynomial x^2 - 5
sage: K1 == K4
False
```

TESTS:

```
sage: QuadraticField(-11, 'a') is QuadraticField(-11, 'a', latex_name='Z')
False
sage: QuadraticField(-11, 'a') is QuadraticField(-11, 'a', latex_name=None)
False
```

sage.rings.number_field.number_field.**is_AbsoluteNumberField**(*x*)

Return True if x is an absolute number field.

EXAMPLES:

```
sage: from sage.rings.number_field.number_field import is_AbsoluteNumberField
sage: is_AbsoluteNumberField(NumberField(x^2+1,'a'))
True
sage: is_AbsoluteNumberField(NumberField([x^3 + 17, x^2+1],'a'))
False
```

The rationals are a number field, but they're not of the absolute number field class.

```
sage: is_AbsoluteNumberField(QQ)
False
```

sage.rings.number_field.number_field.**is_CyclotomicField**(*x*)

Return True if x is a cyclotomic field, i.e., of the special cyclotomic field class. This function does not return True for a number field that just happens to be isomorphic to a cyclotomic field.

EXAMPLES:

```
sage: from sage.rings.number_field.number_field import is_CyclotomicField
sage: is_CyclotomicField(NumberField(x^2 + 1,'zeta4'))
False
sage: is_CyclotomicField(CyclotomicField(4))
True
sage: is_CyclotomicField(CyclotomicField(1))
True
sage: is_CyclotomicField(QQ)
False
sage: is_CyclotomicField(7)
False
```

sage.rings.number_field.number_field.**is_NumberFieldHomsetCodomain**(*codomain*)

Returns whether `codomain` is a valid codomain for a number field homset. This is used by NumberField._Hom_ to determine whether the created homsets should be a `sage.rings.number_field.morphism.NumberFieldHomset`.

EXAMPLES:

This currently accepts any parent (CC, RR, ...) in `Fields`:

```
sage: from sage.rings.number_field.number_field import is_NumberFieldHomsetCodomain
sage: is_NumberFieldHomsetCodomain(QQ)
True
sage: is_NumberFieldHomsetCodomain(NumberField(x^2 + 1, 'x'))
True
sage: is_NumberFieldHomsetCodomain(ZZ)
```

```
False
sage: is_NumberFieldHomsetCodomain(3)
False
sage: is_NumberFieldHomsetCodomain(MatrixSpace(QQ, 2))
False
sage: is_NumberFieldHomsetCodomain(InfinityRing)
False
```

Question: should, for example, QQ-algebras be accepted as well?

Caveat: Gap objects are not (yet) in `Fields`, and therefore not accepted as number field homset codomains:

```
sage: is_NumberFieldHomsetCodomain(gap.Rationals)
False
```

sage.rings.number_field.number_field.**is_QuadraticField**(*x*)

Return True if x is of the quadratic *number* field type.

EXAMPLES:

```
sage: from sage.rings.number_field.number_field import is_QuadraticField
sage: is_QuadraticField(QuadraticField(5,'a'))
True
sage: is_QuadraticField(NumberField(x^2 - 5, 'b'))
True
sage: is_QuadraticField(NumberField(x^3 - 5, 'b'))
False
```

A quadratic field specially refers to a number field, not a finite field:

```
sage: is_QuadraticField(GF(9,'a'))
False
```

sage.rings.number_field.number_field.**is_fundamental_discriminant**(*D*)

Return True if the integer D is a fundamental discriminant, i.e., if $D \cong 0, 1 \pmod 4$, and $D \neq 0, 1$ and either (1) D is square free or (2) we have $D \cong 0 \pmod 4$ with $D/4 \cong 2, 3 \pmod 4$ and $D/4$ square free. These are exactly the discriminants of quadratic fields.

EXAMPLES:

```
sage: [D for D in range(-15,15) if is_fundamental_discriminant(D)]
[-15, -11, -8, -7, -4, -3, 5, 8, 12, 13]
sage: [D for D in range(-15,15) if not is_square(D) and QuadraticField(D,'a').disc() == D]
[-15, -11, -8, -7, -4, -3, 5, 8, 12, 13]
```

sage.rings.number_field.number_field.**proof_flag**(*t*)

Used for easily determining the correct proof flag to use.

Returns t if t is not None, otherwise returns the system-wide proof-flag for number fields (default: True).

EXAMPLES:

```
sage: from sage.rings.number_field.number_field import proof_flag
sage: proof_flag(True)
True
sage: proof_flag(False)
False
sage: proof_flag(None)
True
sage: proof_flag("banana")
'banana'
```

sage.rings.number_field.number_field.**put_natural_embedding_first**(*v*)

Helper function for embeddings() functions for number fields.

INPUT: a list of embeddings of a number field

OUTPUT: None. The list is altered in-place, so that, if possible, the first embedding has been switched with one of the others, so that if there is an embedding which preserves the generator names then it appears first.

EXAMPLES:

```
sage: K.<a> = CyclotomicField(7)
sage: embs = K.embeddings(K)
sage: [e(a) for e in embs] # random - there is no natural sort order
[a, a^2, a^3, a^4, a^5, -a^5 - a^4 - a^3 - a^2 - a - 1]
sage: id = [ e for e in embs if e(a) == a ][0]; id
Ring endomorphism of Cyclotomic Field of order 7 and degree 6
  Defn: a |--> a
sage: permuted_embs = list(embs); permuted_embs.remove(id); permuted_embs.append(id)
sage: [e(a) for e in permuted_embs] # random - but natural map is not first
[a^2, a^3, a^4, a^5, -a^5 - a^4 - a^3 - a^2 - a - 1, a]
sage: permuted_embs[0] != a
True
sage: from sage.rings.number_field.number_field import put_natural_embedding_first
sage: put_natural_embedding_first(permuted_embs)
sage: [e(a) for e in permuted_embs] # random - but natural map is first
[a, a^3, a^4, a^5, -a^5 - a^4 - a^3 - a^2 - a - 1, a^2]
sage: permuted_embs[0] == id
True
```

sage.rings.number_field.number_field.**refine_embedding**(*e*, *prec=None*)

Given an embedding from a number field to either $\mathbf{R}$ or $\mathbf{C}$, returns an equivalent embedding with higher precision.

INPUT:

- e - an embedding of a number field into either RR or CC (with some precision)
- **prec - (default None) the desired precision; if None,** current precision is doubled; if Infinity, the equivalent embedding into either QQbar or AA is returned.

EXAMPLES:

```
sage: from sage.rings.number_field.number_field import refine_embedding
sage: K = CyclotomicField(3)
sage: e10 = K.complex_embedding(10)
sage: e10.codomain().precision()
10
sage: e25 = refine_embedding(e10, prec=25)
sage: e25.codomain().precision()
25
```

An example where we extend a real embedding into AA:

```
sage: K.<a> = NumberField(x^3-2)
sage: K.signature()
(1, 1)
sage: e = K.embeddings(RR)[0]; e
Ring morphism:
From: Number Field in a with defining polynomial x^3 - 2
To:   Real Field with 53 bits of precision
Defn: a |--> 1.25992104989487
sage: e = refine_embedding(e,Infinity); e
```

```
Ring morphism:
From: Number Field in a with defining polynomial x^3 - 2
To:   Algebraic Real Field
Defn: a |--> 1.259921049894873?
```

Now we can obtain arbitrary precision values with no trouble:

```
sage: RealField(150)(e(a))
1.2599210498948731647672106072782283505702515
sage: _^3
2.0000000000000000000000000000000000000000000000
sage: RealField(200)(e(a^2-3*a+7))
4.807637902283579980450073817437623208680738933795329069562 4
```

Complex embeddings can be extended into `QQbar`:

```
sage: e = K.embeddings(CC)[0]; e
Ring morphism:
From: Number Field in a with defining polynomial x^3 - 2
To:   Complex Field with 53 bits of precision
Defn: a |--> -0.62996052494743... - 1.09112363597172*I
sage: e = refine_embedding(e,Infinity); e
Ring morphism:
From: Number Field in a with defining polynomial x^3 - 2
To:   Algebraic Field
Defn: a |--> -0.6299605249474365? - 1.091123635971722?*I
sage: ComplexField(200)(e(a))
-0.62996052494743658238360530363911417528512573235075399004099 - 1.0911236359717214035600726141 8
sage: e(a)^3
2
```

Embeddings into lazy fields work:

```
sage: L = CyclotomicField(7)
sage: x = L.specified_complex_embedding(); x
Generic morphism:
  From: Cyclotomic Field of order 7 and degree 6
  To:   Complex Lazy Field
  Defn: zeta7 -> 0.623489801858734? + 0.781831482468030?*I
sage: refine_embedding(x, 300)
Ring morphism:
  From: Cyclotomic Field of order 7 and degree 6
  To:   Complex Field with 300 bits of precision
  Defn: zeta7 |--> 0.623489801858733530525004884004239810632274730896402105365549439096853652456
sage: refine_embedding(x, infinity)
Ring morphism:
  From: Cyclotomic Field of order 7 and degree 6
  To:   Algebraic Field
  Defn: zeta7 |--> 0.6234898018587335? + 0.7818314824680299?*I
```

When the old embedding is into the real lazy field, then only real embeddings should be considered. See trac ticket #17495:

```
sage: R.<x> = QQ[]
sage: K.<a> = NumberField(x^3 + x - 1, embedding=0.68)
sage: from sage.rings.number_field.number_field import refine_embedding
sage: refine_embedding(K.specified_complex_embedding(), 100)
Ring morphism:
  From: Number Field in a with defining polynomial x^3 + x - 1
  To:   Real Field with 100 bits of precision
```

```
  Defn: a |--> 0.68232780382801932736948373971
sage: refine_embedding(K.specified_complex_embedding(), Infinity)
Ring morphism:
  From: Number Field in a with defining polynomial x^3 + x - 1
  To:   Algebraic Real Field
  Defn: a |--> 0.6823278038280193?
```

CHAPTER

THREE

RELATIVE NUMBER FIELDS

AUTHORS:

- William Stein (2004, 2005): initial version
- Steven Sivek (2006-05-12): added support for relative extensions
- William Stein (2007-09-04): major rewrite and documentation
- Robert Bradshaw (2008-10): specified embeddings into ambient fields
- Nick Alexander (2009-01): modernize coercion implementation
- Robert Harron (2012-08): added is_CM_extension
- Julian Rueth (2014-04-03): absolute number fields are unique parents

This example follows one in the Magma reference manual:

```
sage: K.<y> = NumberField(x^4 - 420*x^2 + 40000)
sage: z = y^5/11; z
420/11*y^3 - 40000/11*y
sage: R.<y> = PolynomialRing(K)
sage: f = y^2 + y + 1
sage: L.<a> = K.extension(f); L
Number Field in a with defining polynomial y^2 + y + 1 over its base field
sage: KL.<b> = NumberField([x^4 - 420*x^2 + 40000, x^2 + x + 1]); KL
Number Field in b0 with defining polynomial x^4 - 420*x^2 + 40000 over its base field
```

We do some arithmetic in a tower of relative number fields:

```
sage: K.<cuberoot2> = NumberField(x^3 - 2)
sage: L.<cuberoot3> = K.extension(x^3 - 3)
sage: S.<sqrt2> = L.extension(x^2 - 2)
sage: S
Number Field in sqrt2 with defining polynomial x^2 - 2 over its base field
sage: sqrt2 * cuberoot3
cuberoot3*sqrt2
sage: (sqrt2 + cuberoot3)^5
(20*cuberoot3^2 + 15*cuberoot3 + 4)*sqrt2 + 3*cuberoot3^2 + 20*cuberoot3 + 60
sage: cuberoot2 + cuberoot3
cuberoot3 + cuberoot2
sage: cuberoot2 + cuberoot3 + sqrt2
sqrt2 + cuberoot3 + cuberoot2
sage: (cuberoot2 + cuberoot3 + sqrt2)^2
(2*cuberoot3 + 2*cuberoot2)*sqrt2 + cuberoot3^2 + 2*cuberoot2*cuberoot3 + cuberoot2^2 + 2
sage: cuberoot2 + sqrt2
sqrt2 + cuberoot2
sage: a = S(cuberoot2); a
```

```
cuberoot2
sage: a.parent()
Number Field in sqrt2 with defining polynomial x^2 - 2 over its base field
```

WARNING: Doing arithmetic in towers of relative fields that depends on canonical coercions is currently VERY SLOW. It is much better to explicitly coerce all elements into a common field, then do arithmetic with them there (which is quite fast).

TESTS:

```
sage: y = polygen(QQ,'y'); K.<beta> = NumberField([y^3 - 3, y^2 - 2])
sage: K(y^10)
27*beta0
sage: beta^10
27*beta0
```

sage.rings.number_field.number_field_rel.**NumberField_extension_v1**(*base_field*, *poly*, *name*, *latex_name*, *canonical_embedding=None*)

This is used in pickling relative fields.

EXAMPLES:

```
sage: from sage.rings.number_field.number_field_rel import NumberField_relative_v1
sage: R.<x> = CyclotomicField(3)[]
sage: NumberField_relative_v1(CyclotomicField(3), x^2 + 7, 'a', 'a')
Number Field in a with defining polynomial x^2 + 7 over its base field
```

class sage.rings.number_field.number_field_rel.**NumberField_relative**(*base*, *polynomial*, *name*, *latex_name=None*, *names=None*, *check=True*, *embedding=None*, *structure=None*)

Bases: sage.rings.number_field.number_field.NumberField_generic

INPUT:

- `base` – the base field
- `polynomial` – a polynomial which must be defined in the ring $K[x]$, where K is the base field.
- `name` – a string, the variable name
- `latex_name` – a string or `None` (default: `None`), variable name for latex printing
- `check` – a boolean (default: `True`), whether to check irreducibility of `polynomial`
- `embedding` – currently not supported, must be `None`
- `structure` – an instance of `structure.NumberFieldStructure` or `None` (default: `None`), provides additional information about this number field, e.g., the absolute number field from which it was created

EXAMPLES:

```
sage: K.<a> = NumberField(x^3 - 2)
sage: t = polygen(K)
sage: L.<b> = K.extension(t^2+t+a); L
Number Field in b with defining polynomial x^2 + x + a over its base field
```

absolute_base_field()

Return the base field of this relative extension, but viewed as an absolute field over $\mathbf{Q}$.

EXAMPLES:

```
sage: K.<a,b,c> = NumberField([x^2 + 2, x^3 + 3, x^3 + 2])
sage: K
Number Field in a with defining polynomial x^2 + 2 over its base field
sage: K.base_field()
Number Field in b with defining polynomial x^3 + 3 over its base field
sage: K.absolute_base_field()[0]
Number Field in a0 with defining polynomial x^9 + 3*x^6 + 165*x^3 + 1
sage: K.base_field().absolute_field('z')
Number Field in z with defining polynomial x^9 + 3*x^6 + 165*x^3 + 1
```

absolute_degree()

The degree of this relative number field over the rational field.

EXAMPLES:

```
sage: K.<a> = NumberFieldTower([x^2 - 17, x^3 - 2])
sage: K.absolute_degree()
6
```

absolute_different()

Return the absolute different of this relative number field L, as an ideal of L. To get the relative different of L/K, use `L.relative_different()`.

EXAMPLES:

```
sage: K.<i> = NumberField(x^2 + 1)
sage: t = K['t'].gen()
sage: L.<b> = K.extension(t^4 - i)
sage: L.absolute_different()
Fractional ideal (8)
```

absolute_discriminant(*v=None*)

Return the absolute discriminant of this relative number field or if `v` is specified, the determinant of the trace pairing on the elements of the list `v`.

INPUT:

- `v` (optional) – list of element of this relative number field.

OUTPUT: Integer if `v` is omitted, and Rational otherwise.

EXAMPLES:

```
sage: K.<i> = NumberField(x^2 + 1)
sage: t = K['t'].gen()
sage: L.<b> = K.extension(t^4 - i)
sage: L.absolute_discriminant()
16777216
sage: L.absolute_discriminant([(b + i)^j for j in range(8)])
61911970349056
```

absolute_field(*names*)

Return an absolute number field K that is isomorphic to this field along with a field-theoretic bijection from self to K and from K to self.

INPUT:

- `names` – string; name of generator of the absolute field

OUTPUT: an absolute number field

Also, `K.structure()` returns `from_K` and `to_K`, where `from_K` is an isomorphism from K to self and `to_K` is an isomorphism from self to K.

EXAMPLES:

```
sage: K.<a,b> = NumberField([x^4 + 3, x^2 + 2]); K
Number Field in a with defining polynomial x^4 + 3 over its base field
sage: L.<xyz> = K.absolute_field(); L
Number Field in xyz with defining polynomial x^8 + 8*x^6 + 30*x^4 - 40*x^2 + 49
sage: L.<c> = K.absolute_field(); L
Number Field in c with defining polynomial x^8 + 8*x^6 + 30*x^4 - 40*x^2 + 49

sage: from_L, to_L = L.structure()
sage: from_L
Isomorphism map:
  From: Number Field in c with defining polynomial x^8 + 8*x^6 + 30*x^4 - 40*x^2 + 49
  To:   Number Field in a with defining polynomial x^4 + 3 over its base field
sage: from_L(c)
a - b
sage: to_L
Isomorphism map:
  From: Number Field in a with defining polynomial x^4 + 3 over its base field
  To:   Number Field in c with defining polynomial x^8 + 8*x^6 + 30*x^4 - 40*x^2 + 49
sage: to_L(a)
-5/182*c^7 - 87/364*c^5 - 185/182*c^3 + 323/364*c
sage: to_L(b)
-5/182*c^7 - 87/364*c^5 - 185/182*c^3 - 41/364*c
sage: to_L(a)^4
-3
sage: to_L(b)^2
-2
```

absolute_generator()

Return the chosen generator over $\mathbf{Q}$ for this relative number field.

EXAMPLES:

```
sage: y = polygen(QQ,'y')
sage: k.<a> = NumberField([y^2 + 2, y^4 + 3])
sage: g = k.absolute_generator(); g
a0 - a1
sage: g.minpoly()
x^2 + 2*a1*x + a1^2 + 2
sage: g.absolute_minpoly()
x^8 + 8*x^6 + 30*x^4 - 40*x^2 + 49
```

absolute_polynomial()

Return the polynomial over $\mathbf{Q}$ that defines this field as an extension of the rational numbers.

EXAMPLES:

```
sage: k.<a, b> = NumberField([x^2 + 1, x^3 + x + 1]); k
Number Field in a with defining polynomial x^2 + 1 over its base field
sage: k.absolute_polynomial()
x^6 + 5*x^4 - 2*x^3 + 4*x^2 + 4*x + 1

sage: k.<a, c> = NumberField([x^2 + 1/3, x^2 + 1/4])
sage: k.absolute_polynomial()
x^4 + 7/6*x^2 + 1/144
sage: k.relative_polynomial()
x^2 + 1/3
```

absolute_polynomial_ntl()

Return defining polynomial of this number field as a pair, an ntl polynomial and a denominator.

This is used mainly to implement some internal arithmetic.

EXAMPLES:

```
sage: NumberField(x^2 + (2/3)*x - 9/17,'a').absolute_polynomial_ntl()
([-27 34 51], 51)
```

absolute_vector_space()

Return vector space over $\mathbf{Q}$ of self and isomorphisms from the vector space to self and in the other direction.

EXAMPLES:

```
sage: K.<a,b> = NumberField([x^3 + 3, x^3 + 2]); K
Number Field in a with defining polynomial x^3 + 3 over its base field
sage: V,from_V,to_V = K.absolute_vector_space(); V
Vector space of dimension 9 over Rational Field
sage: from_V
Isomorphism map:
  From: Vector space of dimension 9 over Rational Field
  To:   Number Field in a with defining polynomial x^3 + 3 over its base field
sage: to_V
Isomorphism map:
  From: Number Field in a with defining polynomial x^3 + 3 over its base field
  To:   Vector space of dimension 9 over Rational Field
sage: c = (a+1)^5; c
7*a^2 - 10*a - 29
sage: to_V(c)
(-29, -712/9, 19712/45, 0, -14/9, 364/45, 0, -4/9, 119/45)
sage: from_V(to_V(c))
7*a^2 - 10*a - 29
sage: from_V(3*to_V(b))
3*b
```

automorphisms()

Compute all Galois automorphisms of self over the base field. This is different than computing the embeddings of self into self; there, automorphisms that do not fix the base field are considered.

EXAMPLES:

```
sage: K.<a, b> = NumberField([x^2 + 10000, x^2 + x + 50]); K
Number Field in a with defining polynomial x^2 + 10000 over its base field
sage: K.automorphisms()
[
Relative number field endomorphism of Number Field in a with defining polynomial x^2 + 10000
  Defn: a |--> a
```

```
        b |--> b,
Relative number field endomorphism of Number Field in a with defining polynomial x^2 + 10000
  Defn: a |--> -a
        b |--> b
]
sage: rho, tau = K.automorphisms()
sage: tau(a)
-a
sage: tau(b) == b
True

sage: L.<b, a> = NumberField([x^2 + x + 50, x^2 + 10000, ]); L
Number Field in b with defining polynomial x^2 + x + 50 over its base field
sage: L.automorphisms()
[
Relative number field endomorphism of Number Field in b with defining polynomial x^2 + x + 5
  Defn: b |--> b
        a |--> a,
Relative number field endomorphism of Number Field in b with defining polynomial x^2 + x + 5
  Defn: b |--> -b - 1
        a |--> a
]
sage: rho, tau = L.automorphisms()
sage: tau(a) == a
True
sage: tau(b)
-b - 1

sage: PQ.<X> = QQ[]
sage: F.<a, b> = NumberField([X^2 - 2, X^2 - 3])
sage: PF.<Y> = F[]
sage: K.<c> = F.extension(Y^2 - (1 + a)*(a + b)*a*b)
sage: K.automorphisms()
[
Relative number field endomorphism of Number Field in c with defining polynomial Y^2 + (-2*b
  Defn: c |--> c
        a |--> a
        b |--> b,
Relative number field endomorphism of Number Field in c with defining polynomial Y^2 + (-2*b
  Defn: c |--> -c
        a |--> a
        b |--> b
]
```

base_field()

Return the base field of this relative number field.

EXAMPLES:

```
sage: k.<a> = NumberField([x^3 + x + 1])
sage: R.<z> = k[]
sage: L.<b> = NumberField(z^3 + a)
sage: L.base_field()
Number Field in a with defining polynomial x^3 + x + 1
sage: L.base_field() is k
True
```

This is very useful because the print representation of a relative field doesn't describe the base field.:

```
sage: L
Number Field in b with defining polynomial z^3 + a over its base field
```

base_ring()

This is exactly the same as base_field.

EXAMPLES:

```
sage: k.<a> = NumberField([x^2 + 1, x^3 + x + 1])
sage: k.base_ring()
Number Field in a1 with defining polynomial x^3 + x + 1
sage: k.base_field()
Number Field in a1 with defining polynomial x^3 + x + 1
```

change_names(*names*)

Return relative number field isomorphic to self but with the given generator names.

INPUT:

- `names` – number of names should be at most the number of generators of self, i.e., the number of steps in the tower of relative fields.

Also, `K.structure()` returns `from_K` and `to_K`, where from_K is an isomorphism from K to self and `to_K` is an isomorphism from self to K.

EXAMPLES:

```
sage: K.<a,b> = NumberField([x^4 + 3, x^2 + 2]); K
Number Field in a with defining polynomial x^4 + 3 over its base field
sage: L.<c,d> = K.change_names()
sage: L
Number Field in c with defining polynomial x^4 + 3 over its base field
sage: L.base_field()
Number Field in d with defining polynomial x^2 + 2
```

An example with a 3-level tower:

```
sage: K.<a,b,c> = NumberField([x^2 + 17, x^2 + x + 1, x^3 - 2]); K
Number Field in a with defining polynomial x^2 + 17 over its base field
sage: L.<m,n,r> = K.change_names()
sage: L
Number Field in m with defining polynomial x^2 + 17 over its base field
sage: L.base_field()
Number Field in n with defining polynomial x^2 + x + 1 over its base field
sage: L.base_field().base_field()
Number Field in r with defining polynomial x^3 - 2
```

And a more complicated example:

```
sage: PQ.<X> = QQ[]
sage: F.<a, b> = NumberField([X^2 - 2, X^2 - 3])
sage: PF.<Y> = F[]
sage: K.<c> = F.extension(Y^2 - (1 + a)*(a + b)*a*b)
sage: L.<m, n, r> = K.change_names(); L
Number Field in m with defining polynomial x^2 + (-2*r - 3)*n - 2*r - 6 over its base field
sage: L.structure()
(Isomorphism given by variable name change map:
  From: Number Field in m with defining polynomial x^2 + (-2*r - 3)*n - 2*r - 6 over its bas
  To:   Number Field in c with defining polynomial Y^2 + (-2*b - 3)*a - 2*b - 6 over its bas
 Isomorphism given by variable name change map:
```

```
From: Number Field in c with defining polynomial Y^2 + (-2*b - 3)*a - 2*b - 6 over its bas
To:   Number Field in m with defining polynomial x^2 + (-2*r - 3)*n - 2*r - 6 over its bas
```

composite_fields(*other*, *names=None*, *both_maps=False*, *preserve_embedding=True*)

List of all possible composite number fields formed from self and other, together with (optionally) embeddings into the compositum; see the documentation for both_maps below.

Since relative fields do not have ambient embeddings, preserve_embedding has no effect. In every case all possible composite number fields are returned.

INPUT:

- `other` - a number field
- `names` - generator name for composite fields
- `both_maps` - (default: False) if True, return quadruples (F, self_into_F, other_into_F, k) such that self_into_F maps self into F, other_into_F maps other into F. For relative number fields k is always None.
- `preserve_embedding` - (default: True) has no effect, but is kept for compatibility with the absolute version of this function. In every case the list of all possible compositums is returned.

OUTPUT:

- `list` - list of the composite fields, possibly with maps.

EXAMPLES:

```
sage: K.<a, b> = NumberField([x^2 + 5, x^2 - 2])
sage: L.<c, d> = NumberField([x^2 + 5, x^2 - 3])
sage: K.composite_fields(L, 'e')
[Number Field in e with defining polynomial x^8 - 24*x^6 + 464*x^4 + 3840*x^2 + 25600]
sage: K.composite_fields(L, 'e', both_maps=True)
[[Number Field in e with defining polynomial x^8 - 24*x^6 + 464*x^4 + 3840*x^2 + 25600,
  Relative number field morphism:
  From: Number Field in a with defining polynomial x^2 + 5 over its base field
 To:   Number Field in e with defining polynomial x^8 - 24*x^6 + 464*x^4 + 3840*x^2 + 25600
  Defn: a |--> -9/66560*e^7 + 11/4160*e^5 - 241/4160*e^3 - 101/104*e
        b |--> -21/166400*e^7 + 73/20800*e^5 - 779/10400*e^3 + 7/260*e,
  Relative number field morphism:
  From: Number Field in c with defining polynomial x^2 + 5 over its base field
  To:   Number Field in e with defining polynomial x^8 - 24*x^6 + 464*x^4 + 3840*x^2 + 25600
  Defn: c |--> -9/66560*e^7 + 11/4160*e^5 - 241/4160*e^3 - 101/104*e
        d |--> -3/25600*e^7 + 7/1600*e^5 - 147/1600*e^3 + 1/40*e,
  None]]
```

defining_polynomial()

Return the defining polynomial of this relative number field.

This is exactly the same as `relative_polynomal()`.

EXAMPLES:

```
sage: C.<z> = CyclotomicField(5)
sage: PC.<X> = C[]
sage: K.<a> = C.extension(X^2 + X + z); K
Number Field in a with defining polynomial X^2 + X + z over its base field
sage: K.defining_polynomial()
X^2 + X + z
```

degree()

The degree, unqualified, of a relative number field is deliberately not implemented, so that a user cannot mistake the absolute degree for the relative degree, or vice versa.

EXAMPLE:

```
sage: K.<a> = NumberFieldTower([x^2 - 17, x^3 - 2])
sage: K.degree()
Traceback (most recent call last):
...
NotImplementedError: For a relative number field you must use relative_degree or absolute_de
```

different()

The different, unqualified, of a relative number field is deliberately not implemented, so that a user cannot mistake the absolute different for the relative different, or vice versa.

EXAMPLE:

```
sage: K.<a> = NumberFieldTower([x^2 + x + 1, x^3 + x + 1])
sage: K.different()
Traceback (most recent call last):
...
NotImplementedError: For a relative number field you must use relative_different or absolute
```

disc()

The discriminant, unqualified, of a relative number field is deliberately not implemented, so that a user cannot mistake the absolute discriminant for the relative discriminant, or vice versa.

EXAMPLE:

```
sage: K.<a> = NumberFieldTower([x^2 + x + 1, x^3 + x + 1])
sage: K.disc()
Traceback (most recent call last):
...
NotImplementedError: For a relative number field you must use relative_discriminant or absol
```

discriminant()

The discriminant, unqualified, of a relative number field is deliberately not implemented, so that a user cannot mistake the absolute discriminant for the relative discriminant, or vice versa.

EXAMPLE:

```
sage: K.<a> = NumberFieldTower([x^2 + x + 1, x^3 + x + 1])
sage: K.discriminant()
Traceback (most recent call last):
...
NotImplementedError: For a relative number field you must use relative_discriminant or absol
```

embeddings(K)

Compute all field embeddings of the relative number field self into the field K (which need not even be a number field, e.g., it could be the complex numbers). This will return an identical result when given K as input again.

If possible, the most natural embedding of self into K is put first in the list.

INPUT:

- K – a field

EXAMPLES:

```
sage: K.<a,b> = NumberField([x^3 - 2, x^2+1])
sage: f = K.embeddings(ComplexField(58)); f
[
Relative number field morphism:
  From: Number Field in a with defining polynomial x^3 - 2 over its base field
  To:   Complex Field with 58 bits of precision
  Defn: a |--> -0.62996052494743676 - 1.0911236359717214*I
        b |--> -1.9428902930940239e-16 + 1.0000000000000000*I,
...
Relative number field morphism:
  From: Number Field in a with defining polynomial x^3 - 2 over its base field
  To:   Complex Field with 58 bits of precision
  Defn: a |--> 1.2599210498948731
        b |--> -0.99999999999999999*I
]
sage: f[0](a)^3
2.0000000000000002 - 8.6389229103644993e-16*I
sage: f[0](b)^2
-1.0000000000000001 - 3.8857805861880480e-16*I
sage: f[0](a+b)
-0.62996052494743693 - 0.091123635971721295*I
```

galois_closure(*names=None*)

Return the absolute number field K that is the Galois closure of this relative number field.

EXAMPLES:

```
sage: K.<a,b> = NumberField([x^4 + 3, x^2 + 2]); K
Number Field in a with defining polynomial x^4 + 3 over its base field
sage: K.galois_closure('c')
Number Field in c with defining polynomial x^16 + 16*x^14 + 28*x^12 + 784*x^10 + 19846*x^8 -
```

galois_group(*type='pari'*, *algorithm='pari'*, *names=None*)

Return the Galois group of the Galois closure of this number field as an abstract group. Note that even though this is an extension L/K, the group will be computed as if it were $L/\mathbf{Q}$.

INPUT:

- `type` - `'pari'` or `'gap'`: type of object to return – a wrapper around a Pari or Gap transitive group object. -
- algorithm - 'pari', 'kash', 'magma' (default: 'pari', except when the degree is >= 12 when 'kash' is tried)

At present much less functionality is available for Galois groups of relative extensions than absolute ones, so try the galois_group method of the corresponding absolute field.

EXAMPLES:

```
sage: x = polygen(QQ)
sage: K.<a> = NumberField(x^2 + 1)
sage: R.<t> = PolynomialRing(K)
sage: L = K.extension(t^5-t+a, 'b')
sage: L.galois_group(type="pari")
Galois group PARI group [240, -1, 22, "S(5)[x]2"] of degree 10 of the Number Field in b with
```

gen(*n=0*)

Return the n'th generator of this relative number field.

EXAMPLES:

```
sage: K.<a,b> = NumberField([x^4 + 3, x^2 + 2]); K
Number Field in a with defining polynomial x^4 + 3 over its base field
sage: K.gens()
(a, b)
sage: K.gen(0)
a
```

gens()

Return the generators of this relative number field.

EXAMPLES:

```
sage: K.<a,b> = NumberField([x^4 + 3, x^2 + 2]); K
Number Field in a with defining polynomial x^4 + 3 over its base field
sage: K.gens()
(a, b)
```

TESTS:

Trivial extensions work like non-trivial ones (trac #2220):

```
sage: NumberField([x^2 - 3, x], 'a').gens()
(a0, 0)
sage: NumberField([x, x^2 - 3], 'a').gens()
(0, a1)
```

is_CM_extension()

Return True is this is a CM extension, i.e. a totally imaginary quadratic extension of a totally real field.

EXAMPLES:

```
sage: F.<a> = NumberField(x^2 - 5)
sage: K.<z> = F.extension(x^2 + 7)
sage: K.is_CM_extension()
True
sage: K = CyclotomicField(7)
sage: K_rel = K.relativize(K.gen() + K.gen()^(-1), 'z')
sage: K_rel.is_CM_extension()
True
sage: F = CyclotomicField(3)
sage: K.<z> = F.extension(x^3 - 2)
sage: K.is_CM_extension()
False
```

A CM field K such that K/F is not a CM extension

```
sage: F.<a> = NumberField(x^2 + 1)
sage: K.<z> = F.extension(x^2 - 3)
sage: K.is_CM_extension()
False
sage: K.is_CM()
True
```

is_absolute()

Returns False, since this is not an absolute field.

EXAMPLES:

```
sage: K.<a,b> = NumberField([x^4 + 3, x^2 + 2]); K
Number Field in a with defining polynomial x^4 + 3 over its base field
sage: K.is_absolute()
```

```
False
sage: K.is_relative()
True
```

is_free(*proof=None*)

Determine whether or not L/K is free (i.e. if $\mathcal{O}_L$ is a free $\mathcal{O}_K$-module).

INPUT:

- `proof` – default: True

EXAMPLES:

```
sage: x = polygen(QQ)
sage: K.<a> = NumberField(x^2+6)
sage: x = polygen(K)
sage: L.<b> = K.extension(x^2 + 3)     ## extend by x^2+3
sage: L.is_free()
False
```

is_galois()

For a relative number field, `is_galois()` is deliberately not implemented, since it is not clear whether this would mean "Galois over **Q**" or "Galois over the given base field". Use either `is_galois_absolute()` or `is_galois_relative()` respectively.

EXAMPLES:

```
sage: k.<a> =NumberField([x^3 - 2, x^2 + x + 1])
sage: k.is_galois()
Traceback (most recent call last):
...
NotImplementedError: For a relative number field L you must use either L.is_galois_relative(
```

is_galois_absolute()

Return True if for this relative extension L/K, L is a Galois extension of **Q**.

EXAMPLE:

```
sage: K.<a> = NumberField(x^3 - 2)
sage: y = polygen(K); L.<b> = K.extension(y^2 - a)
sage: L.is_galois_absolute()
False
```

is_galois_relative()

Return True if for this relative extension L/K, L is a Galois extension of K.

EXAMPLES:

```
sage: K.<a> = NumberField(x^3 - 2)
sage: y = polygen(K)
sage: L.<b> = K.extension(y^2 - a)
sage: L.is_galois_relative()
True
sage: M.<c> = K.extension(y^3 - a)
sage: M.is_galois_relative()
False
```

The following example previously gave the wrong result; see #9390:

```
sage: F.<a, b> = NumberField([x^2 - 2, x^2 - 3])
sage: F.is_galois_relative()
True
```

is_isomorphic_relative(*other*, *base_isom=None*)

For this relative extension L/K and another relative extension M/K, return True if there is a K-linear isomorphism from L to M. More generally, `other` can be a relative extension M/K' with `base_isom` an isomorphism from K to K'.

EXAMPLES:

```
sage: K.<z9> = NumberField(x^6 + x^3 + 1)
sage: R.<z> = PolynomialRing(K)
sage: m1 = 3*z9^4 - 4*z9^3 - 4*z9^2 + 3*z9 - 8
sage: L1 = K.extension(z^2 - m1, 'b1')
sage: G = K.galois_group(); gamma = G.gen()
sage: m2 = (gamma^2)(m1)
sage: L2 = K.extension(z^2 - m2, 'b2')
sage: L1.is_isomorphic_relative(L2)
False
sage: L1.is_isomorphic(L2)
True
sage: L3 = K.extension(z^4 - m1, 'b3')
sage: L1.is_isomorphic_relative(L3)
False
```

If we have two extensions over different, but isomorphic, bases, we can compare them by letting `base_isom` be an isomorphism from self's base field to other's base field:

```
sage: Kcyc.<zeta9> = CyclotomicField(9)
sage: Rcyc.<zcyc> = PolynomialRing(Kcyc)
sage: phi1 = K.hom([zeta9])
sage: m1cyc = phi1(m1)
sage: L1cyc = Kcyc.extension(zcyc^2 - m1cyc, 'b1cyc')
sage: L1.is_isomorphic_relative(L1cyc, base_isom=phi1)
True
sage: L2.is_isomorphic_relative(L1cyc, base_isom=phi1)
False
sage: phi2 = K.hom([phi1((gamma^(-2))(z9))])
sage: L1.is_isomorphic_relative(L1cyc, base_isom=phi2)
False
sage: L2.is_isomorphic_relative(L1cyc, base_isom=phi2)
True
```

Omitting `base_isom` raises a ValueError when the base fields are not identical:

```
sage: L1.is_isomorphic_relative(L1cyc)
Traceback (most recent call last):
...
ValueError: other does not have the same base field as self, so an isomorphism from self's b
```

The parameter `base_isom` can also be used to check if the relative extensions are Galois conjugate:

```
sage: for g in G:
....:     if L1.is_isomorphic_relative(L2, g.as_hom()):
....:         print g.as_hom()
Ring endomorphism of Number Field in z9 with defining polynomial x^6 + x^3 + 1
  Defn: z9 |--> z9^4
```

lift_to_base(*element*)

Lift an element of this extension into the base field if possible, or raise a ValueError if it is not possible.

EXAMPLES:

```
sage: x = polygen(ZZ)
sage: K.<a> = NumberField(x^3 - 2)
sage: R.<y> = K[]
sage: L.<b> = K.extension(y^2 - a)
sage: L.lift_to_base(b^4)
a^2
sage: L.lift_to_base(b^6)
2
sage: L.lift_to_base(355/113)
355/113
sage: L.lift_to_base(b)
Traceback (most recent call last):
...
ValueError: The element b is not in the base field
```

maximal_order(*v=None*)

Return the maximal order, i.e., the ring of integers of this number field.

INPUT:

- v - (default: None) None, a prime, or a list of primes.
  - if v is None, return the maximal order.
  - if v is a prime, return an order that is p-maximal.
  - if v is a list, return an order that is maximal at each prime in the list v.

EXAMPLES:

```
sage: K.<a,b> = NumberField([x^2 + 1, x^2 - 3])
sage: OK = K.maximal_order(); OK.basis()
[1, 1/2*a - 1/2*b, -1/2*b*a + 1/2, a]
sage: charpoly(OK.1)
x^2 + b*x + 1
sage: charpoly(OK.2)
x^2 - x + 1
sage: O2 = K.order([3*a, 2*b])
sage: O2.index_in(OK)
144
```

The following was previously "ridiculously slow"; see trac ticket #4738:

```
sage: K.<a,b> = NumberField([x^4 + 1, x^4 - 3])
sage: K.maximal_order()
Maximal Relative Order in Number Field in a with defining polynomial x^4 + 1 over its base f
```

An example with nontrivial v:

```
sage: L.<a,b> = NumberField([x^2 - 3, x^2 - 5*49])
sage: O3 = L.maximal_order([3])
sage: O3.absolute_discriminant()
8643600
sage: O3.is_maximal()
False
```

ngens()

Return the number of generators of this relative number field.

EXAMPLES:

```
sage: K.<a,b> = NumberField([x^4 + 3, x^2 + 2]); K
Number Field in a with defining polynomial x^4 + 3 over its base field
sage: K.gens()
(a, b)
sage: K.ngens()
2
```

number_of_roots_of_unity()

Return number of roots of unity in this relative field.

EXAMPLES:

```
sage: K.<a, b> = NumberField( [x^2 + x + 1, x^4 + 1] )
sage: K.number_of_roots_of_unity()
24
```

order(**gens*, ***kwds*)

Return the order with given ring generators in the maximal order of this number field.

INPUT:

- `gens` – list of elements of self; if no generators are given, just returns the cardinality of this number field (oo) for consistency.
- `check_is_integral` – bool (default: True), whether to check that each generator is integral.
- `check_rank` – bool (default: True), whether to check that the ring generated by gens is of full rank.
- `allow_subfield` – bool (default: False), if True and the generators do not generate an order, i.e., they generate a subring of smaller rank, instead of raising an error, return an order in a smaller number field.

The check_is_integral and check_rank inputs must be given as explicit keyword arguments.

EXAMPLES:

```
sage: P.<a,b,c> = QQ[2^(1/2), 2^(1/3), 3^(1/2)]
sage: R = P.order([a,b,c]); R
Relative Order in Number Field in sqrt2 with defining polynomial x^2 - 2 over its base field
```

The base ring of an order in a relative extension is still **Z**.:

```
sage: R.base_ring()
Integer Ring
```

One must give enough generators to generate a ring of finite index in the maximal order:

```
sage: P.order([a,b])
Traceback (most recent call last):
...
ValueError: the rank of the span of gens is wrong
```

pari_absolute_base_polynomial()

Return the PARI polynomial defining the absolute base field, in `y`.

EXAMPLES:

```
sage: x = polygen(ZZ)
sage: K.<a, b> = NumberField([x^2 + 2, x^2 + 3]); K
Number Field in a with defining polynomial x^2 + 2 over its base field
sage: K.pari_absolute_base_polynomial()
y^2 + 3
sage: K.pari_absolute_base_polynomial().parent()
```

```
Interface to the PARI C library

sage: z = ZZ['z'].0
sage: K.<a, b, c> = NumberField([z^2 + 2, z^2 + 3, z^2 + 5]); K
Number Field in a with defining polynomial z^2 + 2 over its base field
sage: K.pari_absolute_base_polynomial()
y^4 + 16*y^2 + 4
sage: K.base_field()
Number Field in b with defining polynomial z^2 + 3 over its base field
sage: len(QQ['y'](K.pari_absolute_base_polynomial()).roots(K.base_field()))
4
sage: K.pari_absolute_base_polynomial().parent()
Interface to the PARI C library
```

pari_polynomial(*name='x'*)

PARI polynomial with integer coefficients corresponding to the polynomial that defines this field as an absolute number field.

By default, this is a polynomial in the variable "x". PARI prefers integral polynomials, so we clear the denominator. Therefore, this is NOT the same as simply converting the absolute defining polynomial to PARI.

EXAMPLES:

```
sage: k.<a, c> = NumberField([x^2 + 3, x^2 + 1])
sage: k.pari_polynomial()
x^4 + 8*x^2 + 4
sage: k.pari_polynomial('a')
a^4 + 8*a^2 + 4
sage: k.absolute_polynomial()
x^4 + 8*x^2 + 4
sage: k.relative_polynomial()
x^2 + 3

sage: k.<a, c> = NumberField([x^2 + 1/3, x^2 + 1/4])
sage: k.pari_polynomial()
144*x^4 + 168*x^2 + 1
sage: k.absolute_polynomial()
x^4 + 7/6*x^2 + 1/144
```

pari_relative_polynomial()

Return the PARI relative polynomial associated to this number field.

This is always a polynomial in x and y, suitable for PARI's rnfinit function. Notice that if this is a relative extension of a relative extension, the base field is the absolute base field.

EXAMPLES:

```
sage: k.<i> = NumberField(x^2 + 1)
sage: m.<z> = k.extension(k['w']([i,0,1]))
sage: m
Number Field in z with defining polynomial w^2 + i over its base field
sage: m.pari_relative_polynomial()
Mod(1, y^2 + 1)*x^2 + Mod(y, y^2 + 1)

sage: l.<t> = m.extension(m['t'].0^2 + z)
sage: l.pari_relative_polynomial()
Mod(1, y^4 + 1)*x^2 + Mod(y, y^4 + 1)
```

pari_rnf()

Return the PARI relative number field object associated to this relative extension.

EXAMPLES:

```
sage: k.<a> = NumberField([x^4 + 3, x^2 + 2])
sage: k.pari_rnf()
[x^4 + 3, [[364, -10*x^7 - 87*x^5 - 370*x^3 - 41*x], 1/364], [[108, 0; 0, 108], 3], ...]
```

places (*all_complex=False*, *prec=None*)

Return the collection of all infinite places of self.

By default, this returns the set of real places as homomorphisms into RIF first, followed by a choice of one of each pair of complex conjugate homomorphisms into CIF.

On the other hand, if prec is not None, we simply return places into RealField(prec) and ComplexField(prec) (or RDF, CDF if prec=53).

There is an optional flag all_complex, which defaults to False. If all_complex is True, then the real embeddings are returned as embeddings into CIF instead of RIF.

EXAMPLES:

```
sage: L.<b, c> = NumberFieldTower([x^2 - 5, x^3 + x + 3])
sage: L.places()
[Relative number field morphism:
From: Number Field in b with defining polynomial x^2 - 5 over its base field
To:   Real Field with 106 bits of precision
Defn: b |--> -2.2360679774997896964091736689370
c |--> -1.2134116627622296341321313774260,
Relative number field morphism:
From: Number Field in b with defining polynomial x^2 - 5 over its base field
To:   Real Field with 106 bits of precision
Defn: b |--> 2.2360679774997896964115480053670
c |--> -1.2134116627622296341304924218000,
Relative number field morphism:
From: Number Field in b with defining polynomial x^2 - 5 over its base field
To:   Complex Field with 53 bits of precision
Defn: b |--> -2.23606797749979 ...e-1...*I
c |--> 0.606705831381... - 1.45061224918844*I,
Relative number field morphism:
From: Number Field in b with defining polynomial x^2 - 5 over its base field
To:   Complex Field with 53 bits of precision
Defn: b |--> 2.23606797749979 - 4.44089209850063e-16*I
c |--> 0.606705831381115 - 1.45061224918844*I]
```

polynomial()

For a relative number field, `polynomial()` is deliberately not implemented. Either `relative_polynomial()` or `absolute_polynomial()` must be used.

EXAMPLE:

```
sage: K.<a> = NumberFieldTower([x^2 + x + 1, x^3 + x + 1])
sage: K.polynomial()
Traceback (most recent call last):
...
NotImplementedError: For a relative number field L you must use either L.relative_polynomial
```

relative_degree()

Returns the relative degree of this relative number field.

EXAMPLES:

```
sage: K.<a> = NumberFieldTower([x^2 - 17, x^3 - 2])
sage: K.relative_degree()
2
```

relative_different()

Return the relative different of this extension L/K as an ideal of L. If you want the absolute different of $L/\mathbf{Q}$, use `L.absolute_different()`.

EXAMPLES:

```
sage: K.<i> = NumberField(x^2 + 1)
sage: PK.<t> = K[]
sage: L.<a> = K.extension(t^4  - i)
sage: L.relative_different()
Fractional ideal (4)
```

relative_discriminant()

Return the relative discriminant of this extension L/K as an ideal of K. If you want the (rational) discriminant of $L/\mathbf{Q}$, use e.g. `L.absolute_discriminant()`.

EXAMPLES:

```
sage: K.<i> = NumberField(x^2 + 1)
sage: t = K['t'].gen()
sage: L.<b> = K.extension(t^4 - i)
sage: L.relative_discriminant()
Fractional ideal (256)
sage: PQ.<X> = QQ[]
sage: F.<a, b> = NumberField([X^2 - 2, X^2 - 3])
sage: PF.<Y> = F[]
sage: K.<c> = F.extension(Y^2 - (1 + a)*(a + b)*a*b)
sage: K.relative_discriminant() == F.ideal(4*b)
True
```

relative_polynomial()

Return the defining polynomial of this relative number field over its base field.

EXAMPLES:

```
sage: K.<a> = NumberFieldTower([x^2 + x + 1, x^3 + x + 1])
sage: K.relative_polynomial()
x^2 + x + 1
```

Use absolute polynomial for a polynomial that defines the absolute extension.:

```
sage: K.absolute_polynomial()
x^6 + 3*x^5 + 8*x^4 + 9*x^3 + 7*x^2 + 6*x + 3
```

relative_vector_space()

Return vector space over the base field of self and isomorphisms from the vector space to self and in the other direction.

EXAMPLES:

```
sage: K.<a,b,c> = NumberField([x^2 + 2, x^3 + 2, x^3 + 3]); K
Number Field in a with defining polynomial x^2 + 2 over its base field
sage: V, from_V, to_V = K.relative_vector_space()
sage: from_V(V.0)
1
sage: to_V(K.0)
(0, 1)
```

```
sage: from_V(to_V(K.0))
a
sage: to_V(from_V(V.0))
(1, 0)
sage: to_V(from_V(V.1))
(0, 1)
```

The underlying vector space and maps is cached:

```
sage: W, from_V, to_V = K.relative_vector_space()
sage: V is W
True
```

relativize(*alpha*, *names*)

Given an element in self or an embedding of a subfield into self, return a relative number field K isomorphic to self that is relative over the absolute field $\mathbf{Q}(\alpha)$ or the domain of α, along with isomorphisms from K to self and from self to K.

INPUT:

- `alpha` – an element of self, or an embedding of a subfield into self
- `names` – name of generator for output field K.

OUTPUT: K – a relative number field

Also, `K.structure()` returns `from_K` and `to_K`, where `from_K` is an isomorphism from K to self and `to_K` is an isomorphism from self to K.

EXAMPLES:

```
sage: K.<a,b> = NumberField([x^4 + 3, x^2 + 2]); K
Number Field in a with defining polynomial x^4 + 3 over its base field
sage: L.<z,w> = K.relativize(a^2)
sage: z^2
z^2
sage: w^2
-3
sage: L
Number Field in z with defining polynomial x^4 + (-2*w + 4)*x^2 + 4*w + 1 over its base fiel
sage: L.base_field()
Number Field in w with defining polynomial x^2 + 3
```

Now suppose we have K below L below M:

```
sage: M = NumberField(x^8 + 2, 'a'); M
Number Field in a with defining polynomial x^8 + 2
sage: L, L_into_M, _ = M.subfields(4)[0]; L
Number Field in a0 with defining polynomial x^4 + 2
sage: K, K_into_L, _ = L.subfields(2)[0]; K
Number Field in a0_0 with defining polynomial x^2 + 2
sage: K_into_M = L_into_M * K_into_L

sage: L_over_K = L.relativize(K_into_L, 'c'); L_over_K
Number Field in c0 with defining polynomial x^2 + a0_0 over its base field
sage: L_over_K_to_L, L_to_L_over_K = L_over_K.structure()
sage: M_over_L_over_K = M.relativize(L_into_M * L_over_K_to_L, 'd'); M_over_L_over_K
Number Field in d0 with defining polynomial x^2 + c0 over its base field
sage: M_over_L_over_K.base_field() is L_over_K
True
```

Test relativizing a degree 6 field over its degree 2 and degree 3 subfields, using both an explicit element:

```
sage: K.<a> = NumberField(x^6 + 2); K
Number Field in a with defining polynomial x^6 + 2
sage: K2, K2_into_K, _ = K.subfields(2)[0]; K2
Number Field in a0 with defining polynomial x^2 + 2
sage: K3, K3_into_K, _ = K.subfields(3)[0]; K3
Number Field in a0 with defining polynomial x^3 - 2
```

Here we explicitly relativize over an element of K2 (not the generator):

```
sage: L = K.relativize(K3_into_K, 'b'); L
Number Field in b0 with defining polynomial x^2 + a0 over its base field
sage: L_to_K, K_to_L = L.structure()
sage: L_over_K2 = L.relativize(K_to_L(K2_into_K(K2.gen() + 1)), 'c'); L_over_K2
Number Field in c0 with defining polynomial x^3 - c1 + 1 over its base field
sage: L_over_K2.base_field()
Number Field in c1 with defining polynomial x^2 - 2*x + 3
```

Here we use a morphism to preserve the base field information:

```
sage: K2_into_L = K_to_L * K2_into_K
sage: L_over_K2 = L.relativize(K2_into_L, 'c'); L_over_K2
Number Field in c0 with defining polynomial x^3 - a0 over its base field
sage: L_over_K2.base_field() is K2
True
```

roots_of_unity()

Return all the roots of unity in this relative field, primitive or not.

EXAMPLES:

```
sage: K.<a, b> = NumberField( [x^2 + x + 1, x^4 + 1] )
sage: K.roots_of_unity()[:5]
[b*a, -b^2*a - b^2, b^3, -a, b*a + b]
```

subfields(*degree=0*, *name=None*)

Return all subfields of this relative number field self of the given degree, or of all possible degrees if degree is 0. The subfields are returned as absolute fields together with an embedding into self. For the case of the field itself, the reverse isomorphism is also provided.

EXAMPLES:

```
sage: PQ.<X> = QQ[]
sage: F.<a, b> = NumberField([X^2 - 2, X^2 - 3])
sage: PF.<Y> = F[]
sage: K.<c> = F.extension(Y^2 - (1 + a)*(a + b)*a*b)
sage: K.subfields(2)
[
 (Number Field in c0 with defining polynomial x^2 - 48*x + 288, Ring morphism:
  From: Number Field in c0 with defining polynomial x^2 - 48*x + 288
  To:   Number Field in c with defining polynomial Y^2 + (-2*b - 3)*a - 2*b - 6 over its bas
  Defn: c0 |--> 12*a + 24, None),
 (Number Field in c1 with defining polynomial x^2 - 48*x + 192, Ring morphism:
  From: Number Field in c1 with defining polynomial x^2 - 48*x + 192
  To:   Number Field in c with defining polynomial Y^2 + (-2*b - 3)*a - 2*b - 6 over its bas
  Defn: c1 |--> 8*b*a + 24, None),
 (Number Field in c2 with defining polynomial x^2 - 48*x + 384, Ring morphism:
  From: Number Field in c2 with defining polynomial x^2 - 48*x + 384
  To:   Number Field in c with defining polynomial Y^2 + (-2*b - 3)*a - 2*b - 6 over its bas
  Defn: c2 |--> 8*b + 24, None)
```

```
]
sage: K.subfields(8, 'w')
[
 (Number Field in w0 with defining polynomial x^8 - 24*x^6 + 108*x^4 - 144*x^2 + 36, Ring mo
  From: Number Field in w0 with defining polynomial x^8 - 24*x^6 + 108*x^4 - 144*x^2 + 36
  To:   Number Field in c with defining polynomial Y^2 + (-2*b - 3)*a - 2*b - 6 over its bas
  Defn: w0 |--> c, Relative number field morphism:
  From: Number Field in c with defining polynomial Y^2 + (-2*b - 3)*a - 2*b - 6 over its bas
  To:   Number Field in w0 with defining polynomial x^8 - 24*x^6 + 108*x^4 - 144*x^2 + 36
  Defn: c |--> w0
        a |--> 1/12*w0^6 - 11/6*w0^4 + 11/2*w0^2 - 3
        b |--> -1/24*w0^6 + w0^4 - 17/4*w0^2 + 3)
]
sage: K.subfields(3)
[]
```

uniformizer(*P*, *others='positive'*)

Returns an element of self with valuation 1 at the prime ideal P.

INPUT:

- `self` - a number field
- `P` - a prime ideal of self
- `others` - either "positive" (default), in which case the element will have non-negative valuation at all other primes of self, or "negative", in which case the element will have non-positive valuation at all other primes of self.

Note: When P is principal (e.g. always when self has class number one) the result may or may not be a generator of P!

EXAMPLES:

```
sage: K.<a, b> = NumberField([x^2 + 23, x^2 - 3])
sage: P = K.prime_factors(5)[0]; P
Fractional ideal (5, 1/2*a - b - 5/2)
sage: u = K.uniformizer(P)
sage: u.valuation(P)
1
sage: (P, 1) in K.factor(u)
True
```

vector_space()

For a relative number field, `vector_space()` is deliberately not implemented, so that a user cannot confuse `relative_vector_space()` with `absolute_vector_space()`.

EXAMPLE:

```
sage: K.<a> = NumberFieldTower([x^2 - 17, x^3 - 2])
sage: K.vector_space()
Traceback (most recent call last):
...
NotImplementedError: For a relative number field L you must use either L.relative_vector_spa
```

sage.rings.number_field.number_field_rel.**NumberField_relative_v1**(*base_field*, *poly*, *name*, *latex_name*, *canonical_embedding=None*)

This is used in pickling relative fields.

EXAMPLES:

```
sage: from sage.rings.number_field.number_field_rel import NumberField_relative_v1
sage: R.<x> = CyclotomicField(3)[]
sage: NumberField_relative_v1(CyclotomicField(3), x^2 + 7, 'a', 'a')
Number Field in a with defining polynomial x^2 + 7 over its base field
```

sage.rings.number_field.number_field_rel.**is_RelativeNumberField**(*x*)

Return True if x is a relative number field.

EXAMPLES:

```
sage: from sage.rings.number_field.number_field_rel import is_RelativeNumberField
sage: is_RelativeNumberField(NumberField(x^2+1,'a'))
False
sage: k.<a> = NumberField(x^3 - 2)
sage: l.<b> = k.extension(x^3 - 3); l
Number Field in b with defining polynomial x^3 - 3 over its base field
sage: is_RelativeNumberField(l)
True
sage: is_RelativeNumberField(QQ)
False
```

CHAPTER

FOUR

NUMBER FIELD ELEMENTS

AUTHORS:

- William Stein: version before it got Cython'd
- Joel B. Mohler (2007-03-09): First reimplementation in Cython
- William Stein (2007-09-04): add doctests
- Robert Bradshaw (2007-09-15): specialized classes for relative and absolute elements
- John Cremona (2009-05-15): added support for local and global logarithmic heights.
- Robert Harron (2012-08): conjugate() now works for all fields contained in CM fields

class sage.rings.number_field.number_field_element.**CoordinateFunction**(*alpha*, *W*, *to_V*)

This class provides a callable object which expresses elements in terms of powers of a fixed field generator α.

EXAMPLE:

```
sage: K.<a> = NumberField(x^2 + x + 3)
sage: f = (a + 1).coordinates_in_terms_of_powers(); f
Coordinate function that writes elements in terms of the powers of a + 1
sage: f.__class__
<class sage.rings.number_field.number_field_element.CoordinateFunction at ...>
sage: f(a)
[-1, 1]
sage: f == loads(dumps(f))
True
```

alpha()

EXAMPLE:

```
sage: K.<a> = NumberField(x^3 + 2)
sage: (a + 2).coordinates_in_terms_of_powers().alpha()
a + 2
```

class sage.rings.number_field.number_field_element.**NumberFieldElement**

Bases: sage.structure.element.FieldElement

An element of a number field.

EXAMPLES:

```
sage: k.<a> = NumberField(x^3 + x + 1)
sage: a^3
-a - 1
```

abs (*prec=53*, *i=None*)
Return the absolute value of this element.

If `i` is provided, then the absolute of the i-th embedding is given. Otherwise, if the number field as a defined embedding into $\mathbf{C}$ then the corresponding absolute value is returned and if there is none, it corresponds to the choice `i=0`.

If prec is 53 (the default), then the complex double field is used; otherwise the arbitrary precision (but slow) complex field is used.

INPUT:

- `prec` - (default: 53) integer bits of precision
- `i` - (default:) integer, which embedding to use

EXAMPLES:

```
sage: z = CyclotomicField(7).gen()
sage: abs(z)
1.00000000000000
sage: abs(z^2 + 17*z - 3)
16.0604426799931
sage: K.<a> = NumberField(x^3+17)
sage: abs(a)
2.57128159065824
sage: a.abs(prec=100)
2.5712815906582353554531872087
sage: a.abs(prec=100,i=1)
2.5712815906582353554531872087
sage: a.abs(100, 2)
2.5712815906582353554531872087
```

Here's one where the absolute value depends on the embedding.

```
sage: K.<b> = NumberField(x^2-2)
sage: a = 1 + b
sage: a.abs(i=0)
0.414213562373095
sage: a.abs(i=1)
2.41421356237309
```

Check that trac ticket #16147 is fixed:

```
sage: x = polygen(ZZ)
sage: f = x^3 - x - 1
sage: beta = f.complex_roots()[0]; beta
1.32471795724475
sage: K.<b> = NumberField(f, embedding=beta)
sage: b.abs()
1.32471795724475
```

abs_non_arch (*P*, *prec=None*)
Return the non-archimedean absolute value of this element with respect to the prime P, to the given precision.

INPUT:

- `P` - a prime ideal of the parent of self
- `prec` (int) – desired floating point precision (default: default RealField precision).

OUTPUT:

(real) the non-archimedean absolute value of this element with respect to the prime P, to the given precision. This is the normalised absolute value, so that the underlying prime number p has absolute value $1/p$.

EXAMPLES:

```
sage: K.<a> = NumberField(x^2+5)
sage: [1/K(2).abs_non_arch(P) for P in K.primes_above(2)]
[2.00000000000000]
sage: [1/K(3).abs_non_arch(P) for P in K.primes_above(3)]
[3.00000000000000, 3.00000000000000]
sage: [1/K(5).abs_non_arch(P) for P in K.primes_above(5)]
[5.00000000000000]
```

A relative example:

```
sage: L.<b> = K.extension(x^2-5)
sage: [b.abs_non_arch(P) for P in L.primes_above(b)]
[0.447213595499958, 0.447213595499958]
```

absolute_norm()

Return the absolute norm of this number field element.

EXAMPLES:

```
sage: K1.<a1> = CyclotomicField(11)
sage: K2.<a2> = K1.extension(x^2 - 3)
sage: K3.<a3> = K2.extension(x^2 + 1)
sage: (a1 + a2 + a3).absolute_norm()
1353244757701

sage: QQ(7/5).absolute_norm()
7/5
```

additive_order()

Return the additive order of this element (i.e. infinity if self != 0, 1 if self == 0)

EXAMPLES:

```
sage: K.<u> = NumberField(x^4 - 3*x^2 + 3)
sage: u.additive_order()
+Infinity
sage: K(0).additive_order()
1
sage: K.ring_of_integers().characteristic() # implicit doctest
0
```

charpoly(*var='x'*)

Return the characteristic polynomial of this number field element.

EXAMPLE:

```
sage: K.<a> = NumberField(x^3 + 7)
sage: a.charpoly()
x^3 + 7
sage: K(1).charpoly()
x^3 - 3*x^2 + 3*x - 1
```

complex_embedding(*prec=53*, *i=0*)

Return the i-th embedding of self in the complex numbers, to the given precision.

EXAMPLES:

```
sage: k.<a> = NumberField(x^3 - 2)
sage: a.complex_embedding()
-0.629960524947437 - 1.09112363597172*I
sage: a.complex_embedding(10)
-0.63 - 1.1*I
sage: a.complex_embedding(100)
-0.62996052494743658238360530364 - 1.0911236359717214035600726142*I
sage: a.complex_embedding(20, 1)
-0.62996 + 1.0911*I
sage: a.complex_embedding(20, 2)
1.2599
```

complex_embeddings(*prec=53*)

Return the images of this element in the floating point complex numbers, to the given bits of precision.

INPUT:

- `prec` - integer (default: 53) bits of precision

EXAMPLES:

```
sage: k.<a> = NumberField(x^3 - 2)
sage: a.complex_embeddings()
[-0.629960524947437 - 1.09112363597172*I, -0.629960524947437 + 1.09112363597172*I, 1.259921
sage: a.complex_embeddings(10)
[-0.63 - 1.1*I, -0.63 + 1.1*I, 1.3]
sage: a.complex_embeddings(100)
[-0.62996052494743658238360530364 - 1.0911236359717214035600726142*I, -0.629960524947436582
```

conjugate()

Return the complex conjugate of the number field element.

This is only well-defined for fields contained in CM fields (i.e. for totally real fields and CM fields). Recall that a CM field is a totally imaginary quadratic extension of a totally real field. For other fields, a ValueError is raised.

EXAMPLES:

```
sage: k.<I> = QuadraticField(-1)
sage: I.conjugate()
-I
sage: (I/(1+I)).conjugate()
-1/2*I + 1/2
sage: z6 = CyclotomicField(6).gen(0)
sage: (2*z6).conjugate()
-2*zeta6 + 2
```

The following example now works.

```
sage: F.<b> = NumberField(x^2 - 2)
sage: K.<j> = F.extension(x^2 + 1)
sage: j.conjugate()
-j
```

Raise a ValueError if the field is not contained in a CM field.

```
sage: K.<b> = NumberField(x^3 - 2)
sage: b.conjugate()
Traceback (most recent call last):
...
ValueError: Complex conjugation is only well-defined for fields contained in CM fields.
```

An example of a non-quadratic totally real field.

```
sage: F.<a> = NumberField(x^4 + x^3 - 3*x^2 - x + 1)
sage: a.conjugate()
a
```

An example of a non-cyclotomic CM field.

```
sage: K.<a> = NumberField(x^4 - x^3 + 2*x^2 + x + 1)
sage: a.conjugate()
-1/2*a^3 - a - 1/2
sage: (2*a^2 - 1).conjugate()
a^3 - 2*a^2 - 2
```

coordinates_in_terms_of_powers()

Let α be self. Return a callable object (of type CoordinateFunction) that takes any element of the parent of self in $\mathbf{Q}(\alpha)$ and writes it in terms of the powers of α: $1, \alpha, \alpha^2, ...$

(NOT CACHED).

EXAMPLES:

This function allows us to write elements of a number field in terms of a different generator without having to construct a whole separate number field.

```
sage: y = polygen(QQ,'y'); K.<beta> = NumberField(y^3 - 2); K
Number Field in beta with defining polynomial y^3 - 2
sage: alpha = beta^2 + beta + 1
sage: c = alpha.coordinates_in_terms_of_powers(); c
Coordinate function that writes elements in terms of the powers of beta^2 + beta + 1
sage: c(beta)
[-2, -3, 1]
sage: c(alpha)
[0, 1, 0]
sage: c((1+beta)^5)
[3, 3, 3]
sage: c((1+beta)^10)
[54, 162, 189]
```

This function works even if self only generates a subfield of this number field.

```
sage: k.<a> = NumberField(x^6 - 5)
sage: alpha = a^3
sage: c = alpha.coordinates_in_terms_of_powers()
sage: c((2/3)*a^3 - 5/3)
[-5/3, 2/3]
sage: c
Coordinate function that writes elements in terms of the powers of a^3
sage: c(a)
Traceback (most recent call last):
...
ArithmeticError: vector is not in free module
```

denominator()

Return the denominator of this element, which is by definition the denominator of the corresponding polynomial representation. I.e., elements of number fields are represented as a polynomial (in reduced form) modulo the modulus of the number field, and the denominator is the denominator of this polynomial.

EXAMPLES:

```
sage: K.<z> = CyclotomicField(3)
sage: a = 1/3 + (1/5)*z
```

```
sage: print a.denominator()
15
```

denominator_ideal()

Return the denominator ideal of this number field element.

The denominator ideal of a number field element a is the integral ideal consisting of all elements of the ring of integers R whose product with a is also in R.

See also:

numerator_ideal()

EXAMPLES:

```
sage: K.<a> = NumberField(x^2+5)
sage: b = (1+a)/2
sage: b.norm()
3/2
sage: D = b.denominator_ideal(); D
Fractional ideal (2, a + 1)
sage: D.norm()
2
sage: (1/b).denominator_ideal()
Fractional ideal (3, a + 1)
sage: K(0).denominator_ideal()
Fractional ideal (1)
```

descend_mod_power(*K='QQ'*, *d=2*)

Return a list of elements of the subfield K equal to `self` modulo d'th powers.

INPUT:

- `K` (number field, default QQ) – a subfield of the parent number field L of `self`
- `d` (positive integer, default 2) – an integer at least 2

OUTPUT:

A list, possibly empty, of elements of `K` equal to `self` modulo d'th powers, i.e. the preimages of `self` under the map $K^*/(K^*)^d \to L^*/(L^*)^d$ where L is the parent of `self`. A `ValueError` is raised if K does not embed into L.

ALGORITHM:

All preimages must lie in the Selmer group $K(S,d)$ for a suitable finite set of primes S, which reduces the question to a finite set of possibilities. We may take S to be the set of primes which ramify in L together with those for which the valuation of `self` is not divisible by d.

EXAMPLES:

A relative example:

```
sage: Qi.<i> = QuadraticField(-1)
sage: K.<zeta> = CyclotomicField(8)
sage: f = Qi.embeddings(K)[0]
sage: a = f(2+3*i) * (2-zeta)^2
sage: a.descend_mod_power(Qi,2)
[-3*i - 2, -2*i + 3]
```

An absolute example:

```
sage: K.<zeta> = CyclotomicField(8)
sage: K(1).descend_mod_power(QQ,2)
[1, 2, -1, -2]
sage: a = 17*K.random_element()^2
sage: a.descend_mod_power(QQ,2)
[17, 34, -17, -34]
```

factor()

Return factorization of this element into prime elements and a unit.

OUTPUT:

(Factorization) If all the prime ideals in the support are principal, the output is a Factorization as a product of prime elements raised to appropriate powers, with an appropriate unit factor.

Raise ValueError if the factorization of the ideal (self) contains a non-principal prime ideal.

EXAMPLES:

```
sage: K.<i> = NumberField(x^2+1)
sage: (6*i + 6).factor()
(-i) * (i + 1)^3 * 3
```

In the following example, the class number is 2. If a factorization in prime elements exists, we will find it:

```
sage: K.<a> = NumberField(x^2-10)
sage: factor(169*a + 531)
(-6*a - 19) * (-3*a - 1) * (-2*a + 9)
sage: factor(K(3))
Traceback (most recent call last):
...
ArithmeticError: non-principal ideal in factorization
```

Factorization of 0 is not allowed:

```
sage: K.<i> = QuadraticField(-1)
sage: K(0).factor()
Traceback (most recent call last):
...
ArithmeticError: factorization of 0 is not defined
```

galois_conjugates(*K*)

Return all Gal(Qbar/Q)-conjugates of this number field element in the field K.

EXAMPLES:

In the first example the conjugates are obvious:

```
sage: K.<a> = NumberField(x^2 - 2)
sage: a.galois_conjugates(K)
[a, -a]
sage: K(3).galois_conjugates(K)
[3]
```

In this example the field is not Galois, so we have to pass to an extension to obtain the Galois conjugates.

```
sage: K.<a> = NumberField(x^3 - 2)
sage: c = a.galois_conjugates(K); c
[a]
sage: K.<a> = NumberField(x^3 - 2)
sage: c = a.galois_conjugates(K.galois_closure('a1')); c
[1/18*a1^4, -1/36*a1^4 + 1/2*a1, -1/36*a1^4 - 1/2*a1]
```

```
sage: c[0]^3
2
sage: parent(c[0])
Number Field in a1 with defining polynomial x^6 + 108
sage: parent(c[0]).is_galois()
True
```

There is only one Galois conjugate of $\sqrt[3]{2}$ in $\mathbf{Q}(\sqrt[3]{2})$.

```
sage: a.galois_conjugates(K)
[a]
```

Galois conjugates of $\sqrt[3]{2}$ in the field $\mathbf{Q}(\zeta_3, \sqrt[3]{2})$:

```
sage: L.<a> = CyclotomicField(3).extension(x^3 - 2)
sage: a.galois_conjugates(L)
[a, (-zeta3 - 1)*a, zeta3*a]
```

gcd(*other*)

Return the greatest common divisor of `self` and `other`.

INPUT:

- `self`, `other` – elements of a number field or maximal order.

OUTPUT:

- A generator of the ideal `(self, other)`. If the parent is a number field, this always returns 0 or 1. For maximal orders, this raises `ArithmeticError` if the ideal is not principal.

EXAMPLES:

```
sage: K.<i> = QuadraticField(-1)
sage: (i+1).gcd(2)
1
sage: K(1).gcd(0)
1
sage: K(0).gcd(0)
0
sage: R = K.maximal_order()
sage: R(i+1).gcd(2)
i + 1
```

Non-maximal orders are not supported:

```
sage: R = K.order(2*i)
sage: R(1).gcd(R(4*i))
Traceback (most recent call last):
...
NotImplementedError: gcd() for Order in Number Field in i with defining polynomial x^2 + 1 i
```

The following field has class number 3, but if the ideal `(self, other)` happens to be principal, this still works:

```
sage: K.<a> = NumberField(x^3 - 7)
sage: K.class_number()
3
sage: a.gcd(7)
1
sage: R = K.maximal_order()
sage: R(a).gcd(7)
a
```

```
sage: R(a+1).gcd(2)
Traceback (most recent call last):
...
ArithmeticError: ideal (a + 1, 2) is not principal, gcd is not defined
sage: R(2*a - a^2).gcd(0)
a
```

global_height(*prec=None*)

Returns the absolute logarithmic height of this number field element.

INPUT:

- prec (int) – desired floating point precision (defult: default RealField precision).

OUTPUT:

(real) The absolute logarithmic height of this number field element; that is, the sum of the local heights at all finite and infinite places, scaled by the degree to make the result independent of the parent field.

EXAMPLES:

```
sage: R.<x> = QQ[]
sage: K.<a> = NumberField(x^4+3*x^2-17)
sage: b = a/2
sage: b.global_height()
0.789780699008...
sage: b.global_height(prec=200)
0.78978069900813892060267152032141577237037181070060784564457
```

The global height of an algebraic number is absolute, i.e. it does not depend on the parent field:

```
sage: QQ(6).global_height()
1.79175946922805
sage: K(6).global_height()
1.79175946922805

sage: L.<b> = NumberField((a^2).minpoly())
sage: L.degree()
2
sage: b.global_height() # element of L (degree 2 field)
1.41660667202811
sage: (a^2).global_height() # element of K (degree 4 field)
1.41660667202811
```

And of course every element has the same height as it's inverse:

```
sage: K.<s> = QuadraticField(2)
sage: s.global_height()
0.346573590279973
sage: (1/s).global_height()   #make sure that 11758 is fixed
0.346573590279973
```

global_height_arch(*prec=None*)

Returns the total archimedean component of the height of self.

INPUT:

- prec (int) – desired floating point precision (defult: default RealField precision).

OUTPUT:

(real) The total archimedean component of the height of this number field element; that is, the sum of the local heights at all infinite places.

EXAMPLES:

```
sage: R.<x> = QQ[]
sage: K.<a> = NumberField(x^4+3*x^2-17)
sage: b = a/2
sage: b.global_height_arch()
0.38653407379277...
```

global_height_non_arch(*prec=None*)

Returns the total non-archimedean component of the height of self.

INPUT:

- `prec` (int) – desired floating point precision (default: default RealField precision).

OUTPUT:

(real) The total non-archimedean component of the height of this number field element; that is, the sum of the local heights at all finite places, weighted by the local degrees.

ALGORITHM:

An alternative formula is $\log(d)$ where d is the norm of the denominator ideal; this is used to avoid factorization.

EXAMPLES:

```
sage: R.<x> = QQ[]
sage: K.<a> = NumberField(x^4+3*x^2-17)
sage: b = a/6
sage: b.global_height_non_arch()
7.16703787691222
```

Check that this is equal to the sum of the non-archimedean local heights:

```
sage: [b.local_height(P) for P in b.support()]
[0.000000000000000, 0.693147180559945, 1.09861228866811, 1.09861228866811]
sage: [b.local_height(P, weighted=True) for P in b.support()]
[0.000000000000000, 2.77258872223978, 2.19722457733622, 2.19722457733622]
sage: sum([b.local_height(P,weighted=True) for P in b.support()])
7.16703787691222
```

A relative example:

```
sage: PK.<y> = K[]
sage: L.<c> = NumberField(y^2 + a)
sage: (c/10).global_height_non_arch()
18.4206807439524
```

inverse_mod(*I*)

Returns the inverse of self mod the integral ideal I.

INPUT:

- `I` - may be an ideal of self.parent(), or an element or list of elements of self.parent() generating a nonzero ideal. A ValueError is raised if I is non-integral or zero. A ZeroDivisionError is raised if I + (x) != (1).

NOTE: It's not implemented yet for non-integral elements.

EXAMPLES:

```
sage: k.<a> = NumberField(x^2 + 23)
sage: N = k.ideal(3)
sage: d = 3*a + 1
sage: d.inverse_mod(N)
1

sage: k.<a> = NumberField(x^3 + 11)
sage: d = a + 13
sage: d.inverse_mod(a^2)*d - 1 in k.ideal(a^2)
True
sage: d.inverse_mod((5, a + 1))*d - 1 in k.ideal(5, a + 1)
True
sage: K.<b> = k.extension(x^2 + 3)
sage: b.inverse_mod([37, a - b])
7
sage: 7*b - 1 in K.ideal(37, a - b)
True
sage: b.inverse_mod([37, a - b]).parent() == K
True
```

is_integer()

Test whether this number field element is an integer

EXAMPLES:

```
sage: K.<cbrt3> = NumberField(x^3 - 3)
sage: cbrt3.is_integer()
False
sage: (cbrt3**2 - cbrt3 + 2).is_integer()
False
sage: K(-12).is_integer()
True
sage: K(0).is_integer()
True
sage: K(1/2).is_integer()
False
```

is_integral()

Determine if a number is in the ring of integers of this number field.

EXAMPLES:

```
sage: K.<a> = NumberField(x^2 + 23)
sage: a.is_integral()
True
sage: t = (1+a)/2
sage: t.is_integral()
True
sage: t.minpoly()
x^2 - x + 6
sage: t = a/2
sage: t.is_integral()
False
sage: t.minpoly()
x^2 + 23/4
```

An example in a relative extension:

```
sage: K.<a,b> = NumberField([x^2+1, x^2+3])
sage: (a+b).is_integral()
```

```
True
sage: ((a-b)/2).is_integral()
False
```

is_norm(*L*, *element=False*, *proof=True*)

Determine whether self is the relative norm of an element of L/K, where K is self.parent().

INPUT:

- L – a number field containing K=self.parent()
- element – True or False, whether to also output an element of which self is a norm
- proof – If True, then the output is correct unconditionally. If False, then the output is correct under GRH.

OUTPUT:

If element is False, then the output is a boolean B, which is True if and only if self is the relative norm of an element of L to K. If element is False, then the output is a pair (B, x), where B is as above. If B is True, then x is an element of L such that self == x.norm(K). Otherwise, x is None.

ALGORITHM:

Uses PARI's rnfisnorm. See self._rnfisnorm().

EXAMPLES:

```
sage: K.<beta> = NumberField(x^3+5)
sage: Q.<X> = K[]
sage: L = K.extension(X^2+X+beta, 'gamma')
sage: (beta/2).is_norm(L)
False
sage: beta.is_norm(L)
True
```

With a relative base field:

```
sage: K.<a, b> = NumberField([x^2 - 2, x^2 - 3])
sage: L.<c> = K.extension(x^2 - 5)
sage: (2*a*b).is_norm(L)
True
sage: _, v = (2*b*a).is_norm(L, element=True)
sage: v.norm(K) == 2*a*b
True
```

Non-Galois number fields:

```
sage: K.<a> = NumberField(x^2 + x + 1)
sage: Q.<X> = K[]
sage: L.<b> = NumberField(X^4 + a + 2)
sage: (a/4).is_norm(L)
True
sage: (a/2).is_norm(L)
Traceback (most recent call last):
...
NotImplementedError: is_norm is not implemented unconditionally for norms from non-Galois nu
sage: (a/2).is_norm(L, proof=False)
False

sage: K.<a> = NumberField(x^3 + x + 1)
sage: Q.<X> = K[]
```

```
sage: L.<b> = NumberField(X^4 + a)
sage: t = (-a).is_norm(L, element=True); t
(True, b^3 + 1)
sage: t[1].norm(K)
-a
```

AUTHORS:

- Craig Citro (2008-04-05)
- Marco Streng (2010-12-03)

is_nth_power(*n*)

Return True if `self` is an n'th power in its parent K.

EXAMPLES:

```
sage: K.<a> = NumberField(x^4-7)
sage: K(7).is_nth_power(2)
True
sage: K(7).is_nth_power(4)
True
sage: K(7).is_nth_power(8)
False
sage: K((a-3)^5).is_nth_power(5)
True
```

ALGORITHM: Use PARI to factor x^n - `self` in K.

is_one()

Test whether this number field element is 1.

EXAMPLES:

```
sage: K.<a> = NumberField(x^3 + 3)
sage: K(1).is_one()
True
sage: K(0).is_one()
False
sage: K(-1).is_one()
False
sage: K(1/2).is_one()
False
sage: a.is_one()
False
```

is_rational()

Test whether this number field element is a rational number

EXAMPLES:

```
sage: K.<cbrt3> = NumberField(x^3 - 3)
sage: cbrt3.is_rational()
False
sage: (cbrt3**2 - cbrt3 + 1/2).is_rational()
False
sage: K(-12).is_rational()
True
sage: K(0).is_rational()
True
sage: K(1/2).is_rational()
True
```

is_square(*root=False*)

Return True if self is a square in its parent number field and otherwise return False.

INPUT:

- `root` - if True, also return a square root (or None if self is not a perfect square)

EXAMPLES:

```
sage: m.<b> = NumberField(x^4 - 1789)
sage: b.is_square()
False
sage: c = (2/3*b + 5)^2; c
4/9*b^2 + 20/3*b + 25
sage: c.is_square()
True
sage: c.is_square(True)
(True, 2/3*b + 5)
```

We also test the functional notation.

```
sage: is_square(c, True)
(True, 2/3*b + 5)
sage: is_square(c)
True
sage: is_square(c+1)
False
```

TESTS:

Test that trac ticket #16894 is fixed:

```
sage: K.<a> = QuadraticField(22)
sage: u = K.units()[0]
sage: (u^14).is_square()
True
```

is_totally_positive()

Returns True if self is positive for all real embeddings of its parent number field. We do nothing at complex places, so e.g. any element of a totally complex number field will return True.

EXAMPLES:

```
sage: F.<b> = NumberField(x^3-3*x-1)
sage: b.is_totally_positive()
False
sage: (b^2).is_totally_positive()
True
```

TESTS:

Check that the output is correct even for numbers that are very close to zero (ticket #9596):

```
sage: K.<sqrt2> = QuadraticField(2)
sage: a = 30122754096401; b = 21300003689580
sage: (a/b)^2 > 2
True
sage: (a/b+sqrt2).is_totally_positive()
True
sage: r = RealField(3020)(2).sqrt()*2^3000
sage: a = floor(r)/2^3000
```

```
sage: b = ceil(r)/2^3000
sage: (a+sqrt2).is_totally_positive()
False
sage: (b+sqrt2).is_totally_positive()
True
```

Check that 0 is handled correctly:

```
sage: K.<a> = NumberField(x^5+4*x+1)
sage: K(0).is_totally_positive()
False
```

is_unit()

Return `True` if `self` is a unit in the ring where it is defined.

EXAMPLES:

```
sage: K.<a> = NumberField(x^2 - x - 1)
sage: OK = K.ring_of_integers()
sage: OK(a).is_unit()
True
sage: OK(13).is_unit()
False
sage: K(13).is_unit()
True
```

It also works for relative fields and orders:

```
sage: K.<a,b> = NumberField([x^2 - 3, x^4 + x^3 + x^2 + x + 1])
sage: OK = K.ring_of_integers()
sage: OK(b).is_unit()
True
sage: OK(a).is_unit()
False
sage: a.is_unit()
True
```

list()

Return the list of coefficients of self written in terms of a power basis.

EXAMPLE:

```
sage: K.<a> = NumberField(x^3 - x + 2); ((a + 1)/(a + 2)).list()
[1/4, 1/2, -1/4]
sage: K.<a, b> = NumberField([x^3 - x + 2, x^2 + 23]); ((a + b)/(a + 2)).list()
[3/4*b - 1/2, -1/2*b + 1, 1/4*b - 1/2]
```

local_height(*P*, *prec=None*, *weighted=False*)

Returns the local height of self at a given prime ideal P.

INPUT:

- `P` - a prime ideal of the parent of self
- `prec` (int) – desired floating point precision (defult: default RealField precision).
- `weighted` (bool, default False) – if True, apply local degree weighting.

OUTPUT:

(real) The local height of this number field element at the place P. If `weighted` is True, this is multiplied by the local degree (as required for global heights).

EXAMPLES:

```
sage: R.<x> = QQ[]
sage: K.<a> = NumberField(x^4+3*x^2-17)
sage: P = K.ideal(61).factor()[0][0]
sage: b = 1/(a^2 + 30)
sage: b.local_height(P)
4.11087386417331
sage: b.local_height(P, weighted=True)
8.22174772834662
sage: b.local_height(P, 200)
4.11087386417331124875138910342561474631568174308126106293 74
sage: (b^2).local_height(P)
8.22174772834662
sage: (b^-1).local_height(P)
0.000000000000000
```

A relative example:

```
sage: PK.<y> = K[]
sage: L.<c> = NumberField(y^2 + a)
sage: L(1/4).local_height(L.ideal(2, c-a+1))
1.38629436111989
```

local_height_arch(*i*, *prec=None*, *weighted=False*)

Returns the local height of self at the i'th infinite place.

INPUT:

- **i (int) - an integer in range (r+s) where** (r, s) **is the** signature of the parent field (so $n = r + 2s$ is the degree).
- prec (int) – desired floating point precision (default: default RealField precision).
- weighted (bool, default False) – if True, apply local degree weighting, i.e. double the value for complex places.

OUTPUT:

(real) The archimedean local height of this number field element at the i'th infinite place. If weighted is True, this is multiplied by the local degree (as required for global heights), i.e. 1 for real places and 2 for complex places.

EXAMPLES:

```
sage: R.<x> = QQ[]
sage: K.<a> = NumberField(x^4+3*x^2-17)
sage: [p.codomain() for p in K.places()]
[Real Field with 106 bits of precision,
Real Field with 106 bits of precision,
Complex Field with 53 bits of precision]
sage: [a.local_height_arch(i) for i in range(3)]
[0.53019245457177550833666563897519,
0.53019245457177550833666563897519,
0.886414217456333]
sage: [a.local_height_arch(i, weighted=True) for i in range(3)]
[0.53019245457177550833666563897519,
0.53019245457177550833666563897519,
1.77282843491267]
```

A relative example:

```
sage: L.<b, c> = NumberFieldTower([x^2 - 5, x^3 + x + 3])
sage: [(b + c).local_height_arch(i) for i in range(4)]
[1.2382233907578849118422066174 39,
0.022403472299578757807697469143 91,
0.780028961749618,
1.160489384972 98]
```

matrix(*base=None*)

If base is None, return the matrix of right multiplication by the element on the power basis $1, x, x^2, \ldots, x^{d-1}$ for the number field. Thus the *rows* of this matrix give the images of each of the x^i.

If base is not None, then base must be either a field that embeds in the parent of self or a morphism to the parent of self, in which case this function returns the matrix of multiplication by self on the power basis, where we view the parent field as a field over base.

Specifying base as the base field over which the parent of self is a relative extension is equivalent to base being None

INPUT:

- `base` - field or morphism

EXAMPLES:

Regular number field:

```
sage: K.<a> = NumberField(QQ['x'].0^3 - 5)
sage: M = a.matrix(); M
[0 1 0]
[0 0 1]
[5 0 0]
sage: M.base_ring() is QQ
True
```

Relative number field:

```
sage: L.<b> = K.extension(K['x'].0^2 - 2)
sage: M = b.matrix(); M
[0 1]
[2 0]
sage: M.base_ring() is K
True
```

Absolute number field:

```
sage: M = L.absolute_field('c').gen().matrix(); M
[  0   1   0   0   0   0]
[  0   0   1   0   0   0]
[  0   0   0   1   0   0]
[  0   0   0   0   1   0]
[  0   0   0   0   0   1]
[-17 -60 -12 -10   6   0]
sage: M.base_ring() is QQ
True
```

More complicated relative number field:

```
sage: L.<b> = K.extension(K['x'].0^2 - a); L
Number Field in b with defining polynomial x^2 - a over its base field
sage: M = b.matrix(); M
```

```
[0 1]
[a 0]
sage: M.base_ring() is K
True
```

An example where we explicitly give the subfield or the embedding:

```
sage: K.<a> = NumberField(x^4 + 1); L.<a2> = NumberField(x^2 + 1)
sage: a.matrix(L)
[ 0  1]
[a2  0]
```

Notice that if we compute all embeddings and choose a different one, then the matrix is changed as it should be:

```
sage: v = L.embeddings(K)
sage: a.matrix(v[1])
[  0   1]
[-a2   0]
```

The norm is also changed:

```
sage: a.norm(v[1])
a2
sage: a.norm(v[0])
-a2
```

TESTS:

```
sage: F.<z> = CyclotomicField(5) ; t = 3*z**3 + 4*z**2 + 2
sage: t.matrix(F)
[3*z^3 + 4*z^2 + 2]
sage: x=QQ['x'].gen()
sage: K.<v>=NumberField(x^4 + 514*x^2 + 64321)
sage: R.<r>=NumberField(x^2 + 4*v*x + 5*v^2 + 514)
sage: r.matrix()
[           0            1]
[-5*v^2 - 514         -4*v]
sage: r.matrix(K)
[           0            1]
[-5*v^2 - 514         -4*v]
sage: r.matrix(R)
[r]
sage: foo=R.random_element()
sage: foo.matrix(R) == matrix(1,1,[foo])
True
```

minpoly(*var='x'*)

Return the minimal polynomial of this number field element.

EXAMPLES:

```
sage: K.<a> = NumberField(x^2+3)
sage: a.minpoly('x')
x^2 + 3
sage: R.<X> = K['X']
sage: L.<b> = K.extension(X^2-(22 + a))
sage: b.minpoly('t')
t^2 - a - 22
sage: b.absolute_minpoly('t')
```

```
t^4 - 44*t^2 + 487
sage: b^2 - (22+a)
0
```

multiplicative_order()

Return the multiplicative order of this number field element.

EXAMPLES:

```
sage: K.<z> = CyclotomicField(5)
sage: z.multiplicative_order()
5
sage: (-z).multiplicative_order()
10
sage: (1+z).multiplicative_order()
+Infinity

sage: x = polygen(QQ)
sage: K.<a>=NumberField(x^40 - x^20 + 4)
sage: u = 1/4*a^30 + 1/4*a^10 + 1/2
sage: u.multiplicative_order()
6
sage: a.multiplicative_order()
+Infinity
```

An example in a relative extension:

```
sage: K.<a, b> = NumberField([x^2 + x + 1, x^2 - 3])
sage: z = (a - 1)*b/3
sage: z.multiplicative_order()
12
sage: z^12==1 and z^6!=1 and z^4!=1
True
```

norm(*K=None*)

Return the absolute or relative norm of this number field element.

If K is given then K must be a subfield of the parent L of self, in which case the norm is the relative norm from L to K. In all other cases, the norm is the absolute norm down to QQ.

EXAMPLES:

```
sage: K.<a> = NumberField(x^3 + x^2 + x - 132/7); K
Number Field in a with defining polynomial x^3 + x^2 + x - 132/7
sage: a.norm()
132/7
sage: factor(a.norm())
2^2 * 3 * 7^-1 * 11
sage: K(0).norm()
0
```

Some complicated relatives norms in a tower of number fields.

```
sage: K.<a,b,c> = NumberField([x^2 + 1, x^2 + 3, x^2 + 5])
sage: L = K.base_field(); M = L.base_field()
sage: a.norm()
1
sage: a.norm(L)
1
sage: a.norm(M)
1
```

```
sage: a
a
sage: (a+b+c).norm()
121
sage: (a+b+c).norm(L)
2*c*b - 7
sage: (a+b+c).norm(M)
-11
```

We illustrate that norm is compatible with towers:

```
sage: z = (a+b+c).norm(L); z.norm(M)
-11
```

If we are in an order, the norm is an integer:

```
sage: K.<a> = NumberField(x^3-2)
sage: a.norm().parent()
Rational Field
sage: R = K.ring_of_integers()
sage: R(a).norm().parent()
Integer Ring
```

When the base field is given by an embedding:

```
sage: K.<a> = NumberField(x^4 + 1)
sage: L.<a2> = NumberField(x^2 + 1)
sage: v = L.embeddings(K)
sage: a.norm(v[1])
a2
sage: a.norm(v[0])
-a2
```

TESTS:

```
sage: F.<z> = CyclotomicField(5)
sage: t = 3*z**3 + 4*z**2 + 2
sage: t.norm(F)
3*z^3 + 4*z^2 + 2
```

nth_root(*n*, *all=False*)

Return an n'th root of `self` in its parent K.

EXAMPLES:

```
sage: K.<a> = NumberField(x^4-7)
sage: K(7).nth_root(2)
a^2
sage: K((a-3)^5).nth_root(5)
a - 3
```

ALGORITHM: Use PARI to factor x^n - `self` in K.

numerator_ideal()

Return the numerator ideal of this number field element.

The numerator ideal of a number field element a is the ideal of the ring of integers R obtained by intersecting aR with R.

See also:

`denominator_ideal()`

EXAMPLES:

```
sage: K.<a> = NumberField(x^2+5)
sage: b = (1+a)/2
sage: b.norm()
3/2
sage: N = b.numerator_ideal(); N
Fractional ideal (3, a + 1)
sage: N.norm()
3
sage: (1/b).numerator_ideal()
Fractional ideal (2, a + 1)
sage: K(0).numerator_ideal()
Ideal (0) of Number Field in a with defining polynomial x^2 + 5
```

ord(*P*)

Returns the valuation of self at a given prime ideal P.

INPUT:

- `P` - a prime ideal of the parent of self

Note: The function `ord()` is an alias for `valuation()`.

EXAMPLES:

```
sage: R.<x> = QQ[]
sage: K.<a> = NumberField(x^4+3*x^2-17)
sage: P = K.ideal(61).factor()[0][0]
sage: b = a^2 + 30
sage: b.valuation(P)
1
sage: b.ord(P)
1
sage: type(b.valuation(P))
<type 'sage.rings.integer.Integer'>
```

The function can be applied to elements in relative number fields:

```
sage: L.<b> = K.extension(x^2 - 3)
sage: [L(6).valuation(P) for P in L.primes_above(2)]
[4]
sage: [L(6).valuation(P) for P in L.primes_above(3)]
[2, 2]
```

polynomial(*var='x'*)

Return the underlying polynomial corresponding to this number field element.

The resulting polynomial is currently *not* cached.

EXAMPLES:

```
sage: K.<a> = NumberField(x^5 - x - 1)
sage: f = (-2/3 + 1/3*a)^4; f
1/81*a^4 - 8/81*a^3 + 8/27*a^2 - 32/81*a + 16/81
sage: g = f.polynomial(); g
1/81*x^4 - 8/81*x^3 + 8/27*x^2 - 32/81*x + 16/81
sage: parent(g)
Univariate Polynomial Ring in x over Rational Field
```

Note that the result of this function is not cached (should this be changed?):

```
sage: g is f.polynomial()
False
```

relative_norm()

Return the relative norm of this number field element over the next field down in some tower of number fields.

EXAMPLES:

```
sage: K1.<a1> = CyclotomicField(11)
sage: K2.<a2> = K1.extension(x^2 - 3)
sage: (a1 + a2).relative_norm()
a1^2 - 3
sage: (a1 + a2).relative_norm().relative_norm() == (a1 + a2).absolute_norm()
True

sage: K.<x,y,z> = NumberField([x^2 + 1, x^3 - 3, x^2 - 5])
sage: (x + y + z).relative_norm()
y^2 + 2*z*y + 6
```

residue_symbol(*P*, *m*, *check=True*)

The m-th power residue symbol for an element self and proper ideal P.

$$\left(\frac{\alpha}{\mathbf{P}}\right) \equiv \alpha^{\frac{N(\mathbf{P})-1}{m}} \bmod \mathbf{P}$$

Note: accepts m=1, in which case returns 1

Note: can also be called for an ideal from sage.rings.number_field_ideal.residue_symbol

---

Note: self is coerced into the number field of the ideal P

Note: if m=2, self is an integer, and P is an ideal of a number field of absolute degree 1 (i.e. it is a copy of the rationals), then this calls kronecker_symbol, which is implemented using GMP.

INPUT:

- P - proper ideal of the number field (or an extension)
- m - positive integer

OUTPUT:

- an m-th root of unity in the number field

EXAMPLES:

Quadratic Residue (11 is not a square modulo 17):

```
sage: K.<a> = NumberField(x - 1)
sage: K(11).residue_symbol(K.ideal(17),2)
-1
sage: kronecker_symbol(11,17)
-1
```

The result depends on the number field of the ideal:

```
sage: K.<a> = NumberField(x - 1)
sage: L.<b> = K.extension(x^2 + 1)
sage: K(7).residue_symbol(K.ideal(11),2)
-1
sage: K(7).residue_symbol(L.ideal(11),2)
1
```

Cubic Residue:

```
sage: K.<w> = NumberField(x^2 - x + 1)
sage: (w^2 + 3).residue_symbol(K.ideal(17),3)
-w
```

The field must contain the m-th roots of unity:

```
sage: K.<w> = NumberField(x^2 - x + 1)
sage: (w^2 + 3).residue_symbol(K.ideal(17),5)
Traceback (most recent call last):
...
ValueError: The residue symbol to that power is not defined for the number field
```

sqrt(*all=False*)

Returns the square root of this number in the given number field.

EXAMPLES:

```
sage: K.<a> = NumberField(x^2 - 3)
sage: K(3).sqrt()
a
sage: K(3).sqrt(all=True)
[a, -a]
sage: K(a^10).sqrt()
9*a
sage: K(49).sqrt()
7
sage: K(1+a).sqrt()
Traceback (most recent call last):
...
ValueError: a + 1 not a square in Number Field in a with defining polynomial x^2 - 3
sage: K(0).sqrt()
0
sage: K((7+a)^2).sqrt(all=True)
[a + 7, -a - 7]

sage: K.<a> = CyclotomicField(7)
sage: a.sqrt()
a^4

sage: K.<a> = NumberField(x^5 - x + 1)
sage: (a^4 + a^2 - 3*a + 2).sqrt()
a^3 - a^2
```

ALGORITHM: Use PARI to factor x^2 - `self` in K.

support()

Return the support of this number field element.

OUTPUT: A sorted list of the primes ideals at which this number field element has nonzero valuation. An error is raised if the element is zero.

EXAMPLES:

```
sage: x = ZZ['x'].gen()
sage: F.<t> = NumberField(x^3 - 2)

sage: P5s = F(5).support()
sage: P5s
[Fractional ideal (-t^2 - 1), Fractional ideal (t^2 - 2*t - 1)]
sage: all(5 in P5 for P5 in P5s)
True
sage: all(P5.is_prime() for P5 in P5s)
True
sage: [ P5.norm() for P5 in P5s ]
[5, 25]
```

TESTS:

It doesn't make sense to factor the ideal (0):

```
sage: F(0).support()
Traceback (most recent call last):
...
ArithmeticError: Support of 0 is not defined.
```

trace(*K=None*)

Return the absolute or relative trace of this number field element.

If K is given then K must be a subfield of the parent L of self, in which case the trace is the relative trace from L to K. In all other cases, the trace is the absolute trace down to QQ.

EXAMPLES:

```
sage: K.<a> = NumberField(x^3 -132/7*x^2 + x + 1); K
Number Field in a with defining polynomial x^3 - 132/7*x^2 + x + 1
sage: a.trace()
132/7
sage: (a+1).trace() == a.trace() + 3
True
```

If we are in an order, the trace is an integer:

```
sage: K.<zeta> = CyclotomicField(17)
sage: R = K.ring_of_integers()
sage: R(zeta).trace().parent()
Integer Ring
```

TESTS:

```
sage: F.<z> = CyclotomicField(5) ; t = 3*z**3 + 4*z**2 + 2
sage: t.trace(F)
3*z^3 + 4*z^2 + 2
```

valuation(*P*)

Returns the valuation of self at a given prime ideal P.

INPUT:

- P - a prime ideal of the parent of self

Note: The function `ord()` is an alias for `valuation()`.

EXAMPLES:

```
sage: R.<x> = QQ[]
sage: K.<a> = NumberField(x^4+3*x^2-17)
sage: P = K.ideal(61).factor()[0][0]
sage: b = a^2 + 30
sage: b.valuation(P)
1
sage: b.ord(P)
1
sage: type(b.valuation(P))
<type 'sage.rings.integer.Integer'>
```

The function can be applied to elements in relative number fields:

```
sage: L.<b> = K.extension(x^2 - 3)
sage: [L(6).valuation(P) for P in L.primes_above(2)]
[4]
sage: [L(6).valuation(P) for P in L.primes_above(3)]
[2, 2]
```

vector()

Return vector representation of self in terms of the basis for the ambient number field.

EXAMPLES:

```
sage: K.<a> = NumberField(x^2 + 1)
sage: (2/3*a - 5/6).vector()
(-5/6, 2/3)
sage: (-5/6, 2/3)
(-5/6, 2/3)
sage: O = K.order(2*a)
sage: (O.1).vector()
(0, 2)
sage: K.<a,b> = NumberField([x^2 + 1, x^2 - 3])
sage: (a + b).vector()
(b, 1)
sage: O = K.order([a,b])
sage: (O.1).vector()
(-b, 1)
sage: (O.2).vector()
(1, -b)
```

class sage.rings.number_field.number_field_element.**NumberFieldElement_absolute**

Bases: sage.rings.number_field.number_field_element.NumberFieldElement

absolute_charpoly(*var='x'*, *algorithm=None*)

Return the characteristic polynomial of this element over **Q**.

For the meaning of the optional argument algorithm, see charpoly().

EXAMPLES:

```
sage: x = ZZ['x'].0
sage: K.<a> = NumberField(x^4 + 2, 'a')
sage: a.absolute_charpoly()
x^4 + 2
sage: a.absolute_charpoly('y')
y^4 + 2
sage: (-a^2).absolute_charpoly()
x^4 + 4*x^2 + 4
sage: (-a^2).absolute_minpoly()
```

```
x^2 + 2

sage: a.absolute_charpoly(algorithm='pari') == a.absolute_charpoly(algorithm='sage')
True
```

absolute_minpoly(*var='x', algorithm=None*)

Return the minimal polynomial of this element over $\mathbf{Q}$.

For the meaning of the optional argument algorithm, see `charpoly()`.

EXAMPLES:

```
sage: x = ZZ['x'].0
sage: f = x^10 - 5*x^9 + 15*x^8 - 68*x^7 + 81*x^6 - 221*x^5 + 141*x^4 - 242*x^3 - 13*x^2 - 3
sage: K.<a> = NumberField(f, 'a')
sage: a.absolute_charpoly()
x^10 - 5*x^9 + 15*x^8 - 68*x^7 + 81*x^6 - 221*x^5 + 141*x^4 - 242*x^3 - 13*x^2 - 33*x - 135
sage: a.absolute_charpoly('y')
y^10 - 5*y^9 + 15*y^8 - 68*y^7 + 81*y^6 - 221*y^5 + 141*y^4 - 242*y^3 - 13*y^2 - 33*y - 135
sage: b = -79/9995*a^9 + 52/9995*a^8 + 271/9995*a^7 + 1663/9995*a^6 + 13204/9995*a^5 + 5573/
sage: b.absolute_charpoly()
x^10 + 10*x^9 + 25*x^8 - 80*x^7 - 438*x^6 + 80*x^5 + 2950*x^4 + 1520*x^3 - 10439*x^2 - 5130*
sage: b.absolute_minpoly()
x^5 + 5*x^4 - 40*x^2 - 19*x + 135

sage: b.absolute_minpoly(algorithm='pari') == b.absolute_minpoly(algorithm='sage')
True
```

charpoly(*var='x', algorithm=None*)

The characteristic polynomial of this element, over $\mathbf{Q}$ if self is an element of a field, and over $\mathbf{Z}$ is self is an element of an order.

This is the same as `self.absolute_charpoly` since this is an element of an absolute extension.

The optional argument algorithm controls how the characteristic polynomial is computed: 'pari' uses PARI, 'sage' uses charpoly for Sage matrices. The default value None means that 'pari' is used for small degrees (up to the value of the constant TUNE_CHARPOLY_NF, currently at 25), otherwise 'sage' is used. The constant TUNE_CHARPOLY_NF should give reasonable performance on all architectures; however, if you feel the need to customize it to your own machine, see trac ticket 5213 for a tuning script.

EXAMPLES:

We compute the characteristic polynomial of the cube root of 2.

```
sage: R.<x> = QQ[]
sage: K.<a> = NumberField(x^3-2)
sage: a.charpoly('x')
x^3 - 2
sage: a.charpoly('y').parent()
Univariate Polynomial Ring in y over Rational Field
```

TESTS:

```
sage: R = K.ring_of_integers()
sage: R(a).charpoly()
x^3 - 2
sage: R(a).charpoly().parent()
Univariate Polynomial Ring in x over Integer Ring

sage: R(a).charpoly(algorithm='pari') == R(a).charpoly(algorithm='sage')
True
```

is_real_positive(*min_prec=53*)

Using the `n` method of approximation, return `True` if `self` is a real positive number and `False` otherwise. This method is completely dependent of the embedding used by the `n` method.

The algorithm first checks that `self` is not a strictly complex number. Then if `self` is not zero, by approximation more and more precise, the method answers True if the number is positive. Using $RealInterval$, the result is guaranteed to be correct.

For CyclotomicField, the embedding is the natural one sending $zetan$ on $cos(2 * pi/n)$.

EXAMPLES:

```
sage: K.<a> = CyclotomicField(3)
sage: (a+a^2).is_real_positive()
False
sage: (-a-a^2).is_real_positive()
True
sage: K.<a> = CyclotomicField(1000)
sage: (a+a^(-1)).is_real_positive()
True
sage: K.<a> = CyclotomicField(1009)
sage: d = a^252
sage: (d+d.conjugate()).is_real_positive()
True
sage: d = a^253
sage: (d+d.conjugate()).is_real_positive()
False
sage: K.<a> = QuadraticField(3)
sage: a.is_real_positive()
True
sage: K.<a> = QuadraticField(-3)
sage: a.is_real_positive()
False
sage: (a-a).is_real_positive()
False
```

lift(*var='x'*)

Return an element of QQ[x], where this number field element lives in QQ[x]/(f(x)).

EXAMPLES:

```
sage: K.<a> = QuadraticField(-3)
sage: a.lift()
x
```

list()

Return the list of coefficients of self written in terms of a power basis.

EXAMPLE:

```
sage: K.<z> = CyclotomicField(3)
sage: (2+3/5*z).list()
[2, 3/5]
sage: (5*z).list()
[0, 5]
sage: K(3).list()
[3, 0]
```

minpoly(*var='x'*, *algorithm=None*)

Return the minimal polynomial of this number field element.

For the meaning of the optional argument algorithm, see charpoly().

EXAMPLES:

We compute the characteristic polynomial of cube root of 2.

```
sage: R.<x> = QQ[]
sage: K.<a> = NumberField(x^3-2)
sage: a.minpoly('x')
x^3 - 2
sage: a.minpoly('y').parent()
Univariate Polynomial Ring in y over Rational Field
```

TESTS:

```
sage: R = K.ring_of_integers()
sage: R(a).minpoly()
x^3 - 2
sage: R(a).minpoly().parent()
Univariate Polynomial Ring in x over Integer Ring

sage: R(a).minpoly(algorithm='pari') == R(a).minpoly(algorithm='sage')
True
```

class sage.rings.number_field.number_field_element.**NumberFieldElement_relative**

Bases: sage.rings.number_field.number_field_element.NumberFieldElement

The current relative number field element implementation does everything in terms of absolute polynomials.

All conversions from relative polynomials, lists, vectors, etc should happen in the parent.

absolute_charpoly(*var='x'*, *algorithm=None*)

The characteristic polynomial of this element over $\mathbf{Q}$.

We construct a relative extension and find the characteristic polynomial over $\mathbf{Q}$.

The optional argument algorithm controls how the characteristic polynomial is computed: 'pari' uses PARI, 'sage' uses charpoly for Sage matrices. The default value None means that 'pari' is used for small degrees (up to the value of the constant TUNE_CHARPOLY_NF, currently at 25), otherwise 'sage' is used. The constant TUNE_CHARPOLY_NF should give reasonable performance on all architectures; however, if you feel the need to customize it to your own machine, see trac ticket 5213 for a tuning script.

EXAMPLES:

```
sage: R.<x> = QQ[]
sage: K.<a> = NumberField(x^3-2)
sage: S.<X> = K[]
sage: L.<b> = NumberField(X^3 + 17); L
Number Field in b with defining polynomial X^3 + 17 over its base field
sage: b.absolute_charpoly()
x^9 + 51*x^6 + 867*x^3 + 4913
sage: b.charpoly()(b)
0
sage: a = L.0; a
b
sage: a.absolute_charpoly('x')
x^9 + 51*x^6 + 867*x^3 + 4913
sage: a.absolute_charpoly('y')
y^9 + 51*y^6 + 867*y^3 + 4913
```

```
sage: a.absolute_charpoly(algorithm='pari') == a.absolute_charpoly(algorithm='sage')
True
```

absolute_minpoly(*var='x', algorithm=None*)

Return the minimal polynomial over $\mathbf{Q}$ of this element.

For the meaning of the optional argument `algorithm`, see `absolute_charpoly()`.

EXAMPLES:

```
sage: K.<a, b> = NumberField([x^2 + 2, x^2 + 1000*x + 1])
sage: y = K['y'].0
sage: L.<c> = K.extension(y^2 + a*y + b)
sage: c.absolute_charpoly()
x^8 - 1996*x^6 + 996006*x^4 + 1997996*x^2 + 1
sage: c.absolute_minpoly()
x^8 - 1996*x^6 + 996006*x^4 + 1997996*x^2 + 1
sage: L(a).absolute_charpoly()
x^8 + 8*x^6 + 24*x^4 + 32*x^2 + 16
sage: L(a).absolute_minpoly()
x^2 + 2
sage: L(b).absolute_charpoly()
x^8 + 4000*x^7 + 6000004*x^6 + 4000012000*x^5 + 1000012000006*x^4 + 4000012000*x^3 + 6000004
sage: L(b).absolute_minpoly()
x^2 + 1000*x + 1
```

charpoly(*var='x'*)

The characteristic polynomial of this element over its base field.

EXAMPLES:

```
sage: x = ZZ['x'].0
sage: K.<a, b> = QQ.extension([x^2 + 2, x^5 + 400*x^4 + 11*x^2 + 2])
sage: a.charpoly()
x^2 + 2
sage: b.charpoly()
x^2 - 2*b*x + b^2
sage: b.minpoly()
x - b

sage: K.<a, b> = NumberField([x^2 + 2, x^2 + 1000*x + 1])
sage: y = K['y'].0
sage: L.<c> = K.extension(y^2 + a*y + b)
sage: c.charpoly()
x^2 + a*x + b
sage: c.minpoly()
x^2 + a*x + b
sage: L(a).charpoly()
x^2 - 2*a*x - 2
sage: L(a).minpoly()
x - a
sage: L(b).charpoly()
x^2 - 2*b*x - 1000*b - 1
sage: L(b).minpoly()
x - b
```

lift(*var='x'*)

Return an element of K[x], where this number field element lives in the relative number field K[x]/(f(x)).

EXAMPLES:

```
sage: K.<a> = QuadraticField(-3)
sage: x = polygen(K)
sage: L.<b> = K.extension(x^7 + 5)
sage: u = L(1/2*a + 1/2 + b + (a-9)*b^5)
sage: u.lift()
(a - 9)*x^5 + x + 1/2*a + 1/2
```

list()

Return the list of coefficients of self written in terms of a power basis.

EXAMPLES:

```
sage: K.<a,b> = NumberField([x^3+2, x^2+1])
sage: a.list()
[0, 1, 0]
sage: v = (K.base_field().0 + a)^2 ; v
a^2 + 2*b*a - 1
sage: v.list()
[-1, 2*b, 1]
```

valuation(*P*)

Returns the valuation of self at a given prime ideal P.

INPUT:

- P - a prime ideal of relative number field which is the parent of self

EXAMPLES:

```
sage: K.<a, b, c> = NumberField([x^2 - 2, x^2 - 3, x^2 - 5])
sage: P = K.prime_factors(5)[0]
sage: (2*a + b - c).valuation(P)
1
```

class sage.rings.number_field.number_field_element.**OrderElement_absolute**

Bases: sage.rings.number_field.number_field_element.NumberFieldElement_absolute

Element of an order in an absolute number field.

EXAMPLES:

```
sage: K.<a> = NumberField(x^2 + 1)
sage: O2 = K.order(2*a)
sage: w = O2.1; w
2*a
sage: parent(w)
Order in Number Field in a with defining polynomial x^2 + 1

sage: w.absolute_charpoly()
x^2 + 4
sage: w.absolute_charpoly().parent()
Univariate Polynomial Ring in x over Integer Ring
sage: w.absolute_minpoly()
x^2 + 4
sage: w.absolute_minpoly().parent()
Univariate Polynomial Ring in x over Integer Ring
```

inverse_mod(*I*)

Return an inverse of self modulo the given ideal.

INPUT:

- I - may be an ideal of self.parent(), or an element or list of elements of self.parent() generating a nonzero ideal. A ValueError is raised if I is non-integral or is zero. A ZeroDivisionError is raised if I + (x) != (1).

EXAMPLES:

```
sage: OE = NumberField(x^3 - x + 2, 'w').ring_of_integers()
sage: w = OE.ring_generators()[0]
sage: w.inverse_mod(13*OE)
6*w^2 - 6
sage: w * (w.inverse_mod(13)) - 1 in 13*OE
True
sage: w.inverse_mod(13).parent() == OE
True
sage: w.inverse_mod(2*OE)
Traceback (most recent call last):
...
ZeroDivisionError: w is not invertible modulo Fractional ideal (2)
```

class sage.rings.number_field.number_field_element.**OrderElement_relative**

Bases: sage.rings.number_field.number_field_element.NumberFieldElement_relative

Element of an order in a relative number field.

EXAMPLES:

```
sage: O = EquationOrder([x^2 + x + 1, x^3 - 2],'a,b')
sage: c = O.1; c
(-2*b^2 - 2)*a - 2*b^2 - b
sage: type(c)
<type 'sage.rings.number_field.number_field_element.OrderElement_relative'>
```

absolute_charpoly(*var='x'*)

The absolute characteristic polynomial of this order element over ZZ.

EXAMPLES:

```
sage: x = ZZ['x'].0
sage: K.<a,b> = NumberField([x^2 + 1, x^2 - 3])
sage: OK = K.maximal_order()
sage: _, u, _, v = OK.basis()
sage: t = 2*u - v; t
-b
sage: t.absolute_charpoly()
x^4 - 6*x^2 + 9
sage: t.absolute_minpoly()
x^2 - 3
sage: t.absolute_charpoly().parent()
Univariate Polynomial Ring in x over Integer Ring
```

absolute_minpoly(*var='x'*)

The absolute minimal polynomial of this order element over ZZ.

EXAMPLES:

```
sage: x = ZZ['x'].0
sage: K.<a,b> = NumberField([x^2 + 1, x^2 - 3])
sage: OK = K.maximal_order()
sage: _, u, _, v = OK.basis()
sage: t = 2*u - v; t
-b
sage: t.absolute_charpoly()
```

```
x^4 - 6*x^2 + 9
sage: t.absolute_minpoly()
x^2 - 3
sage: t.absolute_minpoly().parent()
Univariate Polynomial Ring in x over Integer Ring
```

charpoly(*var='x'*)

The characteristic polynomial of this order element over its base ring.

This special implementation works around bug #4738. At this time the base ring of relative order elements is ZZ; it should be the ring of integers of the base field.

EXAMPLES:

```
sage: x = ZZ['x'].0
sage: K.<a,b> = NumberField([x^2 + 1, x^2 - 3])
sage: OK = K.maximal_order(); OK.basis()
[1, 1/2*a - 1/2*b, -1/2*b*a + 1/2, a]
sage: charpoly(OK.1)
x^2 + b*x + 1
sage: charpoly(OK.1).parent()
Univariate Polynomial Ring in x over Maximal Order in Number Field in b with defining polync
sage: [ charpoly(t) for t in OK.basis() ]
[x^2 - 2*x + 1, x^2 + b*x + 1, x^2 - x + 1, x^2 + 1]
```

inverse_mod(*I*)

Return an inverse of self modulo the given ideal.

INPUT:

- `I` - may be an ideal of self.parent(), or an element or list of elements of self.parent() generating a nonzero ideal. A ValueError is raised if I is non-integral or is zero. A ZeroDivisionError is raised if I + (x) != (1).

EXAMPLES:

```
sage: E.<a,b> = NumberField([x^2 - x + 2, x^2 + 1])
sage: OE = E.ring_of_integers()
sage: t = OE(b - a).inverse_mod(17*b)
sage: t*(b - a) - 1 in E.ideal(17*b)
True
sage: t.parent() == OE
True
```

minpoly(*var='x'*)

The minimal polynomial of this order element over its base ring.

This special implementation works around bug #4738. At this time the base ring of relative order elements is ZZ; it should be the ring of integers of the base field.

EXAMPLES:

```
sage: x = ZZ['x'].0
sage: K.<a,b> = NumberField([x^2 + 1, x^2 - 3])
sage: OK = K.maximal_order(); OK.basis()
[1, 1/2*a - 1/2*b, -1/2*b*a + 1/2, a]
sage: minpoly(OK.1)
x^2 + b*x + 1
sage: charpoly(OK.1).parent()
Univariate Polynomial Ring in x over Maximal Order in Number Field in b with defining polync
sage: _, u, _, v = OK.basis()
```

```
sage: t = 2*u - v; t
-b
sage: t.charpoly()
x^2 + 2*b*x + 3
sage: t.minpoly()
x + b

sage: t.absolute_charpoly()
x^4 - 6*x^2 + 9
sage: t.absolute_minpoly()
x^2 - 3
```

sage.rings.number_field.number_field_element.**is_NumberFieldElement**(*x*)

Return True if x is of type NumberFieldElement, i.e., an element of a number field.

EXAMPLES:

```
sage: from sage.rings.number_field.number_field_element import is_NumberFieldElement
sage: is_NumberFieldElement(2)
False
sage: k.<a> = NumberField(x^7 + 17*x + 1)
sage: is_NumberFieldElement(a+1)
True
```

CHAPTER

FIVE

OPTIMIZED QUADRATIC NUMBER FIELD ELEMENTS

This file defines a Cython class `NumberFieldElement_quadratic` to speed up computations in quadratic extensions of **Q**.

AUTHORS:

- Robert Bradshaw (2007-09): Initial version
- David Harvey (2007-10): fix up a few bugs, polish around the edges
- David Loeffler (2009-05): add more documentation and tests
- Vincent Delecroix (2012-07): comparisons for quadratic number fields (trac ticket #13213), abs, floor and ceil functions (trac ticket #13256)

Todo

The `_new()` method should be overridden in this class to copy the `D` and `standard_embedding` attributes

---

class `sage.rings.number_field.number_field_element_quadratic.`**NumberFieldElement_quadratic**

Bases: `sage.rings.number_field.number_field_element.NumberFieldElement_absolute`

A NumberFieldElement_quadratic object gives an efficient representation of an element of a quadratic extension of **Q**.

Elements are represented internally as triples (a, b, c) of integers, where $\gcd(a, b, c) = 1$ and $c > 0$, representing the element $(a + b\sqrt{D})/c$. Note that if the discriminant D is $1 \bmod 4$, integral elements do not necessarily have $c = 1$.

TESTS:

```
sage: from sage.rings.number_field.number_field_element_quadratic import NumberFieldElement_quad
```

We set up some fields:

```
sage: K.<a> = NumberField(x^2+23)
sage: a.parts()
(0, 1)
sage: F.<b> = NumberField(x^2-x+7)
sage: b.parts()
(1/2, 3/2)
```

We construct elements of these fields in various ways - firstly, from polynomials:

```
sage: NumberFieldElement_quadratic(K, x-1)
a - 1
sage: NumberFieldElement_quadratic(F, x-1)
b - 1
```

From triples of Integers:

```
sage: NumberFieldElement_quadratic(K, (1,2,3))
2/3*a + 1/3
sage: NumberFieldElement_quadratic(F, (1,2,3))
4/9*b + 1/9
sage: NumberFieldElement_quadratic(F, (1,2,3)).parts()
(1/3, 2/3)
```

From pairs of Rationals:

```
sage: NumberFieldElement_quadratic(K, (1/2,1/3))
1/3*a + 1/2
sage: NumberFieldElement_quadratic(F, (1/2,1/3))
2/9*b + 7/18
sage: NumberFieldElement_quadratic(F, (1/2,1/3)).parts()
(1/2, 1/3)
```

Direct from Rationals:

```
sage: NumberFieldElement_quadratic(K, 2/3)
2/3
sage: NumberFieldElement_quadratic(F, 2/3)
2/3
```

This checks a bug when converting from lists:

```
sage: w = CyclotomicField(3)([1/2,1])
sage: w == w.__invert__().__invert__()
True
```

ceil()

Returns the ceil.

EXAMPLES:

```
sage: K.<sqrt7> = QuadraticField(7, name='sqrt7')
sage: sqrt7.ceil()
3
sage: (-sqrt7).ceil()
-2
sage: (1022/313*sqrt7 - 14/23).ceil()
9
```

TESTS:

```
sage: K2.<sqrt2> = QuadraticField(2)
sage: K3.<sqrt3> = QuadraticField(3)
sage: K5.<sqrt5> = QuadraticField(5)
sage: for _ in xrange(100):
....:     a = QQ.random_element(1000,20)
....:     b = QQ.random_element(1000,20)
....:     assert ceil(a+b*sqrt(2.)) == ceil(a+b*sqrt2)
....:     assert ceil(a+b*sqrt(3.)) == ceil(a+b*sqrt3)
....:     assert ceil(a+b*sqrt(5.)) == ceil(a+b*sqrt5)

sage: K = QuadraticField(-2)
sage: l = [K(52), K(-3), K(43/12), K(-43/12)]
sage: [x.ceil() for x in l]
[52, -3, 4, -3]
```

charpoly(*var='x'*)

The characteristic polynomial of this element over $\mathbf{Q}$.

EXAMPLES:

```
sage: K.<a> = NumberField(x^2-x+13)
sage: a.charpoly()
x^2 - x + 13
sage: b = 3-a/2
sage: f = b.charpoly(); f
x^2 - 11/2*x + 43/4
sage: f(b)
0
```

continued_fraction()

Return the (finite or ultimately periodic) continued fraction of `self`.

EXAMPLES:

```
sage: K.<sqrt2> = QuadraticField(2)
sage: cf = sqrt2.continued_fraction(); cf
[1; (2)*]
sage: cf.n()
1.41421356237310
sage: sqrt2.n()
1.41421356237310
sage: cf.value()
sqrt2

sage: (sqrt2/3 + 1/4).continued_fraction()
[0; 1, (2, 1, 1, 2, 3, 2, 1, 1, 2, 5, 1, 1, 14, 1, 1, 5)*]
```

continued_fraction_list()

Return the preperiod and the period of the continued fraction expansion of `self`.

EXAMPLES:

```
sage: K.<sqrt2> = QuadraticField(2)
sage: sqrt2.continued_fraction_list()
((1,), (2,))
sage: (1/2+sqrt2/3).continued_fraction_list()
((0, 1, 33), (1, 32))
```

For rational entries a pair of tuples is also returned but the second one is empty:

```
sage: K(123/567).continued_fraction_list()
((0, 4, 1, 1, 1, 1, 3, 2), ())
```

denominator()

Return the denominator of self. This is the LCM of the denominators of the coefficients of self, and thus it may well be > 1 even when the element is an algebraic integer.

EXAMPLES:

```
sage: K.<a> = NumberField(x^2+x+41)
sage: a.denominator()
1
sage: b = (2*a+1)/6
sage: b.denominator()
6
sage: K(1).denominator()
1
sage: K(1/2).denominator()
```

```
2
sage: K(0).denominator()
1

sage: K.<a> = NumberField(x^2 - 5)
sage: b = (a + 1)/2
sage: b.denominator()
2
sage: b.is_integral()
True
```

floor()

Returns the floor of x.

EXAMPLES:

```
sage: K.<sqrt2> = QuadraticField(2,name='sqrt2')
sage: sqrt2.floor()
1
sage: (-sqrt2).floor()
-2
sage: (13/197 + 3702/123*sqrt2).floor()
42
sage: (13/197-3702/123*sqrt2).floor()
-43
```

TESTS:

```
sage: K2.<sqrt2> = QuadraticField(2)
sage: K3.<sqrt3> = QuadraticField(3)
sage: K5.<sqrt5> = QuadraticField(5)
sage: for _ in xrange(100):
....:     a = QQ.random_element(1000,20)
....:     b = QQ.random_element(1000,20)
....:     assert floor(a+b*sqrt(2.)) == floor(a+b*sqrt2)
....:     assert floor(a+b*sqrt(3.)) == floor(a+b*sqrt3)
....:     assert floor(a+b*sqrt(5.)) == floor(a+b*sqrt5)

sage: K = QuadraticField(-2)
sage: l = [K(52), K(-3), K(43/12), K(-43/12)]
sage: [x.floor() for x in l]
[52, -3, 3, -4]
```

galois_conjugate()

Return the image of this element under action of the nontrivial element of the Galois group of this field.

EXAMPLES:

```
sage: K.<a> = QuadraticField(23)
sage: a.galois_conjugate()
-a

sage: K.<a> = NumberField(x^2 - 5*x + 1)
sage: a.galois_conjugate()
-a + 5
sage: b = 5*a + 1/3
sage: b.galois_conjugate()
-5*a + 76/3
sage: b.norm() ==  b * b.galois_conjugate()
True
```

```
sage: b.trace() ==  b + b.galois_conjugate()
True
```

imag()

Returns the imaginary part of self.

EXAMPLES:

```
sage: K.<sqrt2> = QuadraticField(2)
sage: sqrt2.imag()
0
sage: parent(sqrt2.imag())
Rational Field

sage: K.<i> = QuadraticField(-1)
sage: i.imag()
1
sage: parent(i.imag())
Rational Field

sage: K.<a> = NumberField(x^2 + x + 1, embedding=CDF.0)
sage: a.imag()
1/2*sqrt3
sage: a.real()
-1/2
sage: SR(a)
1/2*I*sqrt(3) - 1/2
sage: bool(I*a.imag() + a.real() == a)
True
```

TESTS:

```
sage: K.<a> = QuadraticField(-9, embedding=-CDF.0)
sage: a.imag()
-3
sage: parent(a.imag())
Rational Field
```

is_integer()

Check whether this number field element is an integer.

EXAMPLES:

```
sage: K.<sqrt3> = QuadraticField(3)
sage: sqrt3.is_integer()
False
sage: (sqrt3-1/2).is_integer()
False
sage: K(0).is_integer()
True
sage: K(-12).is_integer()
True
sage: K(1/3).is_integer()
False
```

is_integral()

Returns whether this element is an algebraic integer.

TESTS:

```
sage: K.<a> = QuadraticField(-1)
sage: a.is_integral()
True
sage: K(1).is_integral()
True
sage: K(1/2).is_integral()
False
sage: K(a/2).is_integral()
False
sage: K((a+1)/2).is_integral()
False
sage: K(a/3).is_integral()
False

sage: K.<a> = QuadraticField(-3)
sage: a.is_integral()
True
sage: K(1).is_integral()
True
sage: K(1/2).is_integral()
False
sage: K(a/2).is_integral()
False
sage: ((a+1)/2).is_integral()
True
```

is_one()

Check whether this number field element is 1.

EXAMPLES:

```
sage: K = QuadraticField(-2)
sage: K(1).is_one()
True
sage: K(-1).is_one()
False
sage: K(2).is_one()
False
sage: K(0).is_one()
False
sage: K(1/2).is_one()
False
sage: K.gen().is_one()
False
```

is_rational()

Check whether this number field element is a rational number.

EXAMPLES:

```
sage: K.<sqrt3> = QuadraticField(3)
sage: sqrt3.is_rational()
False
sage: (sqrt3-1/2).is_rational()
False
sage: K(0).is_rational()
True
sage: K(-12).is_rational()
True
sage: K(1/3).is_rational()
```

```
True
```

minpoly(*var='x'*)

The minimal polynomial of this element over **Q**.

INPUT:

- `var` – the minimal polynomial is defined over a polynomial ring in a variable with this name. If not specified this defaults to `x`.

EXAMPLES:

```
sage: K.<a> = NumberField(x^2+13)
sage: a.minpoly()
x^2 + 13
sage: a.minpoly('T')
T^2 + 13
sage: (a+1/2-a).minpoly()
x - 1/2
```

norm(*K=None*)

Return the norm of self. If the second argument is None, this is the norm down to **Q**. Otherwise, return the norm down to K (which had better be either **Q** or this number field).

EXAMPLES:

```
sage: K.<a> = NumberField(x^2-x+3)
sage: a.norm()
3
sage: a.matrix()
[ 0  1]
[-3  1]
sage: K.<a> = NumberField(x^2+5)
sage: (1+a).norm()
6
```

The norm is multiplicative:

```
sage: K.<a> = NumberField(x^2-3)
sage: a.norm()
-3
sage: K(3).norm()
9
sage: (3*a).norm()
-27
```

We test that the optional argument is handled sensibly:

```
sage: (3*a).norm(QQ)
-27
sage: (3*a).norm(K)
3*a
sage: (3*a).norm(CyclotomicField(3))
Traceback (most recent call last):
...
ValueError: no way to embed L into parent's base ring K
```

numerator()

Return self*self.denominator().

EXAMPLES:

```
sage: K.<a> = NumberField(x^2+x+41)
sage: b = (2*a+1)/6
sage: b.denominator()
6
sage: b.numerator()
2*a + 1
```

parts()

This function returns a pair of rationals a and b such that self $= a + b\sqrt{D}$.

This is much closer to the internal storage format of the elements than the polynomial representation coefficients (the output of `self.list()`), unless the generator with which this number field was constructed was equal to $\sqrt{D}$. See the last example below.

EXAMPLES:

```
sage: K.<a> = NumberField(x^2-13)
sage: K.discriminant()
13
sage: a.parts()
(0, 1)
sage: (a/2-4).parts()
(-4, 1/2)
sage: K.<a> = NumberField(x^2-7)
sage: K.discriminant()
28
sage: a.parts()
(0, 1)
sage: K.<a> = NumberField(x^2-x+7)
sage: a.parts()
(1/2, 3/2)
sage: a._coefficients()
[0, 1]
```

real()

Returns the real part of self, which is either self (if self lives it a totally real field) or a rational number.

EXAMPLES:

```
sage: K.<sqrt2> = QuadraticField(2)
sage: sqrt2.real()
sqrt2
sage: K.<a> = QuadraticField(-3)
sage: a.real()
0
sage: (a + 1/2).real()
1/2
sage: K.<a> = NumberField(x^2 + x + 1)
sage: a.real()
-1/2
sage: parent(a.real())
Rational Field
sage: K.<i> = QuadraticField(-1)
sage: i.real()
0
```

sign()

Returns the sign of self (0 if zero, +1 if positive and -1 if negative).

EXAMPLES:

```
sage: K.<sqrt2> = QuadraticField(2, name='sqrt2')
sage: K(0).sign()
0
sage: sqrt2.sign()
1
sage: (sqrt2+1).sign()
1
sage: (sqrt2-1).sign()
1
sage: (sqrt2-2).sign()
-1
sage: (-sqrt2).sign()
-1
sage: (-sqrt2+1).sign()
-1
sage: (-sqrt2+2).sign()
1

sage: K.<a> = QuadraticField(2, embedding=-1.4142)
sage: K(0).sign()
0
sage: a.sign()
-1
sage: (a+1).sign()
-1
sage: (a+2).sign()
1
sage: (a-1).sign()
-1
sage: (-a).sign()
1
sage: (-a-1).sign()
1
sage: (-a-2).sign()
-1

sage: K.<b> = NumberField(x^2 + 2*x + 7, 'b', embedding=CC(-1,-sqrt(6)))
sage: b.sign()
Traceback (most recent call last):
...
ValueError: a complex number has no sign!
sage: K(1).sign()
1
sage: K(0).sign()
0
sage: K(-2/3).sign()
-1
```

trace()

EXAMPLES:

```
sage: K.<a> = NumberField(x^2+x+41)
sage: a.trace()
-1
sage: a.matrix()
[  0   1]
[-41  -1]
```

The trace is additive:

```
sage: K.<a> = NumberField(x^2+7)
sage: (a+1).trace()
2
sage: K(3).trace()
6
sage: (a+4).trace()
8
sage: (a/3+1).trace()
2
```

class sage.rings.number_field.number_field_element_quadratic.**OrderElement_quadratic**

Bases: sage.rings.number_field.number_field_element_quadratic.NumberFieldElement_quadrati

Element of an order in a quadratic field.

EXAMPLES:

```
sage: K.<a> = NumberField(x^2 + 1)
sage: O2 = K.order(2*a)
sage: w = O2.1; w
2*a
sage: parent(w)
Order in Number Field in a with defining polynomial x^2 + 1
```

charpoly(*var='x'*)

The characteristic polynomial of this element, which is over **Z** because this element is an algebraic integer.

EXAMPLES:

```
sage: K.<a> = NumberField(x^2 - 5)
sage: R = K.ring_of_integers()
sage: b = R((5+a)/2)
sage: f = b.charpoly('x'); f
x^2 - 5*x + 5
sage: f.parent()
Univariate Polynomial Ring in x over Integer Ring
sage: f(b)
0
```

inverse_mod(*I*)

Return an inverse of self modulo the given ideal.

INPUT:

- `I` - may be an ideal of self.parent(), or an element or list of elements of self.parent() generating a nonzero ideal. A ValueError is raised if I is non-integral or is zero. A ZeroDivisionError is raised if I + (x) != (1).

EXAMPLES:

```
sage: OE = QuadraticField(-7, 's').ring_of_integers()
sage: w = OE.ring_generators()[0]
sage: w.inverse_mod(13) == 6*w - 6
True
sage: w*(6*w - 6) - 1
-13
sage: w.inverse_mod(13).parent() == OE
True
sage: w.inverse_mod(2*OE)
Traceback (most recent call last):
```

```
...
ZeroDivisionError: 1/2*s + 1/2 is not invertible modulo Fractional ideal (2)
```

minpoly(*var='x'*)

The minimal polynomial of this element over **Z**.

EXAMPLES:

```
sage: K.<a> = NumberField(x^2 + 163)
sage: R = K.ring_of_integers()
sage: f = R(a).minpoly('x'); f
x^2 + 163
sage: f.parent()
Univariate Polynomial Ring in x over Integer Ring
sage: R(5).minpoly()
x - 5
```

norm()

The norm of an element of the ring of integers is an Integer.

EXAMPLES:

```
sage: K.<a> = NumberField(x^2 + 3)
sage: O2 = K.order(2*a)
sage: w = O2.gen(1); w
2*a
sage: w.norm()
12
sage: parent(w.norm())
Integer Ring
```

trace()

The trace of an element of the ring of integers is an Integer.

EXAMPLES:

```
sage: K.<a> = NumberField(x^2 - 5)
sage: R = K.ring_of_integers()
sage: b = R((1+a)/2)
sage: b.trace()
1
sage: parent(b.trace())
Integer Ring
```

class sage.rings.number_field.number_field_element_quadratic.**Q_to_quadratic_field_element**

Bases: sage.categories.morphism.Morphism

Morphism that coerces from rationals to elements of a quadratic number field K.

class sage.rings.number_field.number_field_element_quadratic.**Z_to_quadratic_field_element**

Bases: sage.categories.morphism.Morphism

Morphism that coerces from integers to elements of a quadratic number field K.

CHAPTER

SIX

ORDERS IN NUMBER FIELDS

AUTHORS:

- William Stein and Robert Bradshaw (2007-09): initial version

EXAMPLES:

We define an absolute order:

```
sage: K.<a> = NumberField(x^2 + 1); O = K.order(2*a)
sage: O.basis()
[1, 2*a]
```

We compute a basis for an order in a relative extension that is generated by 2 elements:

```
sage: K.<a,b> = NumberField([x^2 + 1, x^2 - 3]); O = K.order([3*a,2*b])
sage: O.basis()
[1, 3*a - 2*b, -6*b*a + 6, 3*a]
```

We compute a maximal order of a degree 10 field:

```
sage: K.<a> = NumberField((x+1)^10 + 17)
sage: K.maximal_order()
Maximal Order in Number Field in a with defining polynomial x^10 + 10*x^9 + 45*x^8 + 120*x^7 + 210*x^
```

We compute a suborder, which has index a power of 17 in the maximal order:

```
sage: O = K.order(17*a); O
Order in Number Field in a with defining polynomial x^10 + 10*x^9 + 45*x^8 + 120*x^7 + 210*x^6 + 252*
sage: m = O.index_in(K.maximal_order()); m
2345316516532778891166559194441622630463080918373248225 7
sage: factor(m)
17^45
```

class sage.rings.number_field.order.**AbsoluteOrder**(*K*, *module_rep*, *is_maximal=None*, *check=True*)

Bases: sage.rings.number_field.order.Order

EXAMPLES:

```
sage: from sage.rings.number_field.order import *
sage: x = polygen(QQ)
sage: K.<a> = NumberField(x^3+2)
sage: V, from_v, to_v = K.vector_space()
sage: M = span([to_v(a^2), to_v(a), to_v(1)],ZZ)
sage: O = AbsoluteOrder(K, M); O
Order in Number Field in a with defining polynomial x^3 + 2
```

```
sage: M = span([to_v(a^2), to_v(a), to_v(2)],ZZ)
sage: O = AbsoluteOrder(K, M); O
Traceback (most recent call last):
...
ValueError: 1 is not in the span of the module, hence not an order.

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

Quadratic elements have a special optimized type:

absolute_discriminant()

Return the discriminant of this order.

EXAMPLES:

```
sage: K.<a> = NumberField(x^8 + x^3 - 13*x + 26)
sage: O = K.maximal_order()
sage: factor(O.discriminant())
3 * 11 * 13^2 * 613 * 1575917857
sage: L = K.order(13*a^2)
sage: factor(L.discriminant())
3^3 * 5^2 * 11 * 13^60 * 613 * 733^2 * 1575917857
sage: factor(L.index_in(O))
3 * 5 * 13^29 * 733
sage: L.discriminant() / O.discriminant() == L.index_in(O)^2
True
```

absolute_order()

Return the absolute order associated to this order, which is just this order again since this is an absolute order.

EXAMPLES:

```
sage: K.<a> = NumberField(x^3 + 2)
sage: O1 = K.order(a); O1
Order in Number Field in a with defining polynomial x^3 + 2
sage: O1.absolute_order() is O1
True
```

basis()

Return the basis over $\mathbf{Z}$ for this order.

EXAMPLES:

```
sage: k.<c> = NumberField(x^3 + x^2 + 1)
sage: O = k.maximal_order(); O
Maximal Order in Number Field in c with defining polynomial x^3 + x^2 + 1
sage: O.basis()
[1, c, c^2]
```

The basis is an immutable sequence:

```
sage: type(O.basis())
<class 'sage.structure.sequence.Sequence_generic'>
```

The generator functionality uses the basis method:

```
sage: O.0
1
sage: O.1
```

```
c
sage: O.gens()
[1, c, c^2]
sage: O.ngens()
3
```

change_names(*names*)

Return a new order isomorphic to this one in the number field with given variable names.

EXAMPLES:

```
sage: R = EquationOrder(x^3 + x + 1, 'alpha'); R
Order in Number Field in alpha with defining polynomial x^3 + x + 1
sage: R.basis()
[1, alpha, alpha^2]
sage: S = R.change_names('gamma'); S
Order in Number Field in gamma with defining polynomial x^3 + x + 1
sage: S.basis()
[1, gamma, gamma^2]
```

discriminant()

Return the discriminant of this order.

EXAMPLES:

```
sage: K.<a> = NumberField(x^8 + x^3 - 13*x + 26)
sage: O = K.maximal_order()
sage: factor(O.discriminant())
3 * 11 * 13^2 * 613 * 1575917857
sage: L = K.order(13*a^2)
sage: factor(L.discriminant())
3^3 * 5^2 * 11 * 13^60 * 613 * 733^2 * 1575917857
sage: factor(L.index_in(O))
3 * 5 * 13^29 * 733
sage: L.discriminant() / O.discriminant() == L.index_in(O)^2
True
```

index_in(*other*)

Return the index of self in other. This is a lattice index, so it is a rational number if self isn't contained in other.

INPUT:

- `other` – another absolute order with the same ambient number field.

OUTPUT:

a rational number

EXAMPLES:

```
sage: k.<i> = NumberField(x^2 + 1)
sage: O1 = k.order(i)
sage: O5 = k.order(5*i)
sage: O5.index_in(O1)
5

sage: k.<a> = NumberField(x^3 + x^2 - 2*x+8)
sage: o = k.maximal_order()
sage: o
Maximal Order in Number Field in a with defining polynomial x^3 + x^2 - 2*x + 8
sage: O1 = k.order(a); O1
```

```
Order in Number Field in a with defining polynomial x^3 + x^2 - 2*x + 8
sage: O1.index_in(o)
2
sage: O2 = k.order(1+2*a); O2
Order in Number Field in a with defining polynomial x^3 + x^2 - 2*x + 8
sage: O1.basis()
[1, a, a^2]
sage: O2.basis()
[1, 2*a, 4*a^2]
sage: o.index_in(O2)
1/16
```

intersection(*other*)

Return the intersection of this order with another order.

EXAMPLES:

```
sage: k.<i> = NumberField(x^2 + 1)
sage: O6 = k.order(6*i)
sage: O9 = k.order(9*i)
sage: O6.basis()
[1, 6*i]
sage: O9.basis()
[1, 9*i]
sage: O6.intersection(O9).basis()
[1, 18*i]
sage: (O6 & O9).basis()
[1, 18*i]
sage: (O6 + O9).basis()
[1, 3*i]
```

module()

Returns the underlying free module corresponding to this order, embedded in the vector space corresponding to the ambient number field.

EXAMPLES:

```
sage: k.<a> = NumberField(x^3 + x + 3)
sage: m = k.order(3*a); m
Order in Number Field in a with defining polynomial x^3 + x + 3
sage: m.module()
Free module of degree 3 and rank 3 over Integer Ring
Echelon basis matrix:
[1 0 0]
[0 3 0]
[0 0 9]
```

sage.rings.number_field.order.**EquationOrder**(*f*, *names*)

Return the equation order generated by a root of the irreducible polynomial f or list of polynomials f (to construct a relative equation order).

IMPORTANT: Note that the generators of the returned order need *not* be roots of f, since the generators of an order are – in Sage – module generators.

EXAMPLES:

```
sage: O.<a,b> = EquationOrder([x^2+1, x^2+2])
sage: O
Relative Order in Number Field in a with defining polynomial x^2 + 1 over its base field
sage: O.0
```

```
-b*a - 1
sage: O.1
-3*a + 2*b
```

Of course the input polynomial must be integral:

```
sage: R = EquationOrder(x^3 + x + 1/3, 'alpha'); R
Traceback (most recent call last):
...
ValueError: each generator must be integral

sage: R = EquationOrder( [x^3 + x + 1, x^2 + 1/2], 'alpha'); R
Traceback (most recent call last):
...
ValueError: each generator must be integral
```

class sage.rings.number_field.order.**Order**(*K*, *is_maximal*)

Bases: sage.rings.ring.IntegralDomain

An order in a number field.

An order is a subring of the number field that has **Z**-rank equal to the degree of the number field over **Q**.

EXAMPLES:

```
sage: K.<theta> = NumberField(x^4 + x + 17)
sage: K.maximal_order()
Maximal Order in Number Field in theta with defining polynomial x^4 + x + 17
sage: R = K.order(17*theta); R
Order in Number Field in theta with defining polynomial x^4 + x + 17
sage: R.basis()
[1, 17*theta, 289*theta^2, 4913*theta^3]
sage: R = K.order(17*theta, 13*theta); R
Order in Number Field in theta with defining polynomial x^4 + x + 17
sage: R.basis()
[1, theta, theta^2, theta^3]
sage: R = K.order([34*theta, 17*theta + 17]); R
Order in Number Field in theta with defining polynomial x^4 + x + 17

sage: K.<b> = NumberField(x^4 + x^2 + 2)
sage: (b^2).charpoly().factor()
(x^2 + x + 2)^2
sage: K.order(b^2)
Traceback (most recent call last):
...
ValueError: the rank of the span of gens is wrong
```

absolute_degree()

Returns the absolute degree of this order, ie the degree of this order over **Z**.

EXAMPLES:

```
sage: K.<a> = NumberField(x^3 + 2)
sage: O = K.maximal_order()
sage: O.absolute_degree()
3
```

ambient()

Return the ambient number field that contains self.

This is the same as self.number_field() and self.fraction_field()

EXAMPLES:

```
sage: k.<z> = NumberField(x^2 - 389)
sage: o = k.order(389*z + 1)
sage: o
Order in Number Field in z with defining polynomial x^2 - 389
sage: o.basis()
[1, 389*z]
sage: o.ambient()
Number Field in z with defining polynomial x^2 - 389
```

basis()

Return a basis over **Z** of this order.

EXAMPLES:

```
sage: K.<a> = NumberField(x^3 + x^2 - 16*x + 16)
sage: O = K.maximal_order(); O
Maximal Order in Number Field in a with defining polynomial x^3 + x^2 - 16*x + 16
sage: O.basis()
[1, 1/4*a^2 + 1/4*a, a^2]
```

class_group(*proof=None*, *names='c'*)

Return the class group of this order.

(Currently only implemented for the maximal order.)

EXAMPLES:

```
sage: k.<a> = NumberField(x^2 + 5077)
sage: O = k.maximal_order(); O
Maximal Order in Number Field in a with defining polynomial x^2 + 5077
sage: O.class_group()
Class group of order 22 with structure C22 of Number Field in a with defining polynomial x^2
```

class_number(*proof=None*)

Return the class number of this order.

EXAMPLES:

```
sage: ZZ[2^(1/3)].class_number()
1
sage: QQ[sqrt(-23)].maximal_order().class_number()
3
```

Note that non-maximal orders aren't supported yet:

```
sage: ZZ[3*sqrt(-3)].class_number()
Traceback (most recent call last):
...
NotImplementedError: computation of class numbers of non-maximal orders is not implemented
```

coordinates(*x*)

Returns the coordinate vector of x with respect to this order.

INPUT:

- x – an element of the number field of this order.

OUTPUT:

A vector of length n (the degree of the field) giving the coordinates of x with respect to the integral basis of the order. In general this will be a vector of rationals; it will consist of integers if and only if x is in the

order.

AUTHOR: John Cremona 2008-11-15

ALGORITHM:

Uses linear algebra. The change-of-basis matrix is cached. Provides simpler implementations for `_contains_()`, `is_integral()` and `smallest_integer()`.

EXAMPLES:

```
sage: K.<i> = QuadraticField(-1)
sage: OK = K.ring_of_integers()
sage: OK_basis = OK.basis(); OK_basis
[1, i]
sage: a = 23-14*i
sage: acoords = OK.coordinates(a); acoords
(23, -14)
sage: sum([OK_basis[j]*acoords[j] for j in range(2)]) == a
True
sage: OK.coordinates((120+340*i)/8)
(15, 85/2)

sage: O = K.order(3*i)
sage: O.is_maximal()
False
sage: O.index_in(OK)
3
sage: acoords = O.coordinates(a); acoords
(23, -14/3)
sage: sum([O.basis()[j]*acoords[j] for j in range(2)]) == a
True
```

degree()

Return the degree of this order, which is the rank of this order as a $\mathbf{Z}$-module.

EXAMPLES:

```
sage: k.<c> = NumberField(x^3 + x^2 - 2*x+8)
sage: o = k.maximal_order()
sage: o.degree()
3
sage: o.rank()
3
```

fraction_field()

Return the fraction field of this order, which is the ambient number field.

EXAMPLES:

```
sage: K.<b> = NumberField(x^4 + 17*x^2 + 17)
sage: O = K.order(17*b); O
Order in Number Field in b with defining polynomial x^4 + 17*x^2 + 17
sage: O.fraction_field()
Number Field in b with defining polynomial x^4 + 17*x^2 + 17
```

fractional_ideal(**args*, ***kwds*)

Return the fractional ideal of the maximal order with given generators.

EXAMPLES:

```
sage: K.<a> = NumberField(x^2 + 2)
sage: R = K.maximal_order()
```

```
sage: R.fractional_ideal(2/3 + 7*a, a)
Fractional ideal (1/3*a)
```

free_module()

Return the free $\mathbf{Z}$-module contained in the vector space associated to the ambient number field, that corresponds to this ideal.

EXAMPLES:

```
sage: K.<a> = NumberField(x^3 + x^2 - 2*x + 8)
sage: O = K.maximal_order(); O.basis()
[1, 1/2*a^2 + 1/2*a, a^2]
sage: O.free_module()
Free module of degree 3 and rank 3 over Integer Ring
User basis matrix:
[  1   0   0]
[  0 1/2 1/2]
[  0   0   1]
```

An example in a relative extension. Notice that the module is a $\mathbf{Z}$-module in the absolute_field associated to the relative field:

```
sage: K.<a,b> = NumberField([x^2 + 1, x^2 + 2])
sage: O = K.maximal_order(); O.basis()
[(-3/2*b - 5)*a + 7/2*b - 2, -3*a + 2*b, -2*b*a - 3, -7*a + 5*b]
sage: O.free_module()
Free module of degree 4 and rank 4 over Integer Ring
User basis matrix:
[1/4 1/4 3/4 3/4]
[  0 1/2   0 1/2]
[  0   0   1   0]
[  0   0   0   1]
```

gen(i)

Return i'th module generator of this order.

EXAMPLES:

```
sage: K.<c> = NumberField(x^3 + 2*x + 17)
sage: O = K.maximal_order(); O
Maximal Order in Number Field in c with defining polynomial x^3 + 2*x + 17
sage: O.basis()
[1, c, c^2]
sage: O.gen(1)
c
sage: O.gen(2)
c^2
sage: O.gen(5)
Traceback (most recent call last):
...
IndexError: no 5th generator
sage: O.gen(-1)
Traceback (most recent call last):
...
IndexError: no -1th generator
```

gens()

Return a list of the module generators of this order.

Note: For a (much smaller) list of ring generators use `ring_generators()`.

EXAMPLES:

```
sage: K.<a> = NumberField(x^3 + x^2 - 2*x + 8)
sage: O = K.maximal_order()
sage: O.gens()
[1, 1/2*a^2 + 1/2*a, a^2]
```

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

Return the integral ideal with given generators.

EXAMPLES:

```
sage: K.<a> = NumberField(x^2 + 7)
sage: R = K.maximal_order()
sage: R.ideal(2/3 + 7*a, a)
Traceback (most recent call last):
...
ValueError: ideal must be integral; use fractional_ideal to create a non-integral ideal.
sage: R.ideal(7*a, 77 + 28*a)
Fractional ideal (7)
sage: R = K.order(4*a)
sage: R.ideal(8)
Traceback (most recent call last):
...
NotImplementedError: ideals of non-maximal orders not yet supported.
```

This function is called implicitly below:

```
sage: R = EquationOrder(x^2 + 2, 'a'); R
Order in Number Field in a with defining polynomial x^2 + 2
sage: (3,15)*R
Fractional ideal (3)
```

The zero ideal is handled properly:

```
sage: R.ideal(0)
Ideal (0) of Number Field in a with defining polynomial x^2 + 2
```

integral_closure()

Return the integral closure of this order.

EXAMPLES:

```
sage: K.<a> = QuadraticField(5)
sage: O2 = K.order(2*a); O2
Order in Number Field in a with defining polynomial x^2 - 5
sage: O2.integral_closure()
Maximal Order in Number Field in a with defining polynomial x^2 - 5
sage: OK = K.maximal_order()
sage: OK is OK.integral_closure()
True
```

is_field(*proof=True*)

Return False (because an order is never a field).

EXAMPLES:

```
sage: L.<alpha> = NumberField(x**4 - x**2 + 7)
sage: O = L.maximal_order() ; O.is_field()
False
```

```
sage: CyclotomicField(12).ring_of_integers().is_field()
False
```

is_integrally_closed()

Return True if this ring is integrally closed, i.e., is equal to the maximal order.

EXAMPLES:

```
sage: K.<a> = NumberField(x^2 + 189*x + 394)
sage: R = K.order(2*a)
sage: R.is_integrally_closed()
False
sage: R
Order in Number Field in a with defining polynomial x^2 + 189*x + 394
sage: S = K.maximal_order(); S
Maximal Order in Number Field in a with defining polynomial x^2 + 189*x + 394
sage: S.is_integrally_closed()
True
```

is_maximal()

Returns True if this is the maximal order.

EXAMPLE:

```
sage: k.<i> = NumberField(x^2 + 1)
sage: O3 = k.order(3*i); O5 = k.order(5*i); Ok = k.maximal_order(); Osum = O3 + O5
sage: Osum.is_maximal()
True
sage: O3.is_maximal()
False
sage: O5.is_maximal()
False
sage: Ok.is_maximal()
True
```

An example involving a relative order::

```
sage: K.<a,b> = NumberField([x^2 + 1, x^2 - 3]); O = K.order([3*a,2*b]); O
Relative Order in Number Field in a with defining polynomial x^2 + 1 over its base field
sage: O.is_maximal()
False
```

is_noetherian()

Return True (because orders are always Noetherian)

EXAMPLES:

```
sage: L.<alpha> = NumberField(x**4 - x**2 + 7)
sage: O = L.maximal_order() ; O.is_noetherian()
True
sage: E.<w> = NumberField(x^2 - x + 2)
sage: OE = E.ring_of_integers(); OE.is_noetherian()
True
```

is_suborder(*other*)

Return True if self and other are both orders in the same ambient number field and self is a subset of other.

EXAMPLES:

```
sage: W.<i> = NumberField(x^2 + 1)
sage: O5 = W.order(5*i)
```

```
sage: O10 = W.order(10*i)
sage: O15 = W.order(15*i)
sage: O15.is_suborder(O5)
True
sage: O5.is_suborder(O15)
False
sage: O10.is_suborder(O15)
False
```

We create another isomorphic but different field:

```
sage: W2.<j> = NumberField(x^2 + 1)
sage: P5 = W2.order(5*j)
```

This is False because the ambient number fields are not equal.:

```
sage: O5.is_suborder(P5)
False
```

We create a field that contains (in no natural way!) W, and of course again is_suborder returns False:

```
sage: K.<z> = NumberField(x^4 + 1)
sage: M = K.order(5*z)
sage: O5.is_suborder(M)
False
```

krull_dimension()

Return the Krull dimension of this order, which is 1.

EXAMPLES:

```
sage: K.<a> = QuadraticField(5)
sage: OK = K.maximal_order()
sage: OK.krull_dimension()
1
sage: O2 = K.order(2*a)
sage: O2.krull_dimension()
1
```

ngens()

Return the number of module generators of this order.

EXAMPLES:

```
sage: K.<a> = NumberField(x^3 + x^2 - 2*x + 8)
sage: O = K.maximal_order()
sage: O.ngens()
3
```

number_field()

Return the number field of this order, which is the ambient number field that this order is embedded in.

EXAMPLES:

```
sage: K.<b> = NumberField(x^4 + x^2 + 2)
sage: O = K.order(2*b); O
Order in Number Field in b with defining polynomial x^4 + x^2 + 2
sage: O.basis()
[1, 2*b, 4*b^2, 8*b^3]
sage: O.number_field()
Number Field in b with defining polynomial x^4 + x^2 + 2
```

```
sage: O.number_field() is K
True
```

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

Return a random element of this order.

INPUT:

- `args, kwds` – parameters passed to the random integer function. See the documentation for `ZZ.random_element()` for details.

OUTPUT:

A random element of this order, computed as a random **Z**-linear combination of the basis.

EXAMPLES:

```
sage: K.<a> = NumberField(x^3 + 2)
sage: OK = K.ring_of_integers()
sage: OK.random_element() # random output
-2*a^2 - a - 2
sage: OK.random_element(distribution="uniform") # random output
-a^2 - 1
sage: OK.random_element(-10,10) # random output
-10*a^2 - 9*a - 2
sage: K.order(a).random_element() # random output
a^2 - a - 3

sage: K.<z> = CyclotomicField(17)
sage: OK = K.ring_of_integers()
sage: OK.random_element() # random output
z^15 - z^11 - z^10 - 4*z^9 + z^8 + 2*z^7 + z^6 - 2*z^5 - z^4 - 445*z^3 - 2*z^2 - 15*z - 2
sage: OK.random_element().is_integral()
True
sage: OK.random_element().parent() is OK
True
```

A relative example:

```
sage: K.<a, b> = NumberField([x^2 + 2, x^2 + 1000*x + 1])
sage: OK = K.ring_of_integers()
sage: OK.random_element() # random output
(42221/2*b + 61/2)*a + 7037384*b + 7041
sage: OK.random_element().is_integral() # random output
True
sage: OK.random_element().parent() is OK # random output
True
```

An example in a non-maximal order:

```
sage: K.<a> = QuadraticField(-3)
sage: R = K.ring_of_integers()
sage: A = K.order(a)
sage: A.index_in(R)
2
sage: R.random_element() # random output
-39/2*a - 1/2
sage: A.random_element() # random output
2*a - 1
sage: A.random_element().is_integral()
True
```

```
sage: A.random_element().parent() is A
True
```

rank()

Return the rank of this order, which is the rank of the underlying $\mathbf{Z}$-module, or the degree of the ambient number field that contains this order.

This is a synonym for `degree()`.

EXAMPLES:

```
sage: k.<c> = NumberField(x^5 + x^2 + 1)
sage: o = k.maximal_order(); o
Maximal Order in Number Field in c with defining polynomial x^5 + x^2 + 1
sage: o.rank()
5
```

residue_field(*prime*, *names=None*, *check=False*)

Return the residue field of this order at a given prime, ie O/pO.

INPUT:

- `prime` – a prime ideal of the maximal order in this number field.
- `names` – the name of the variable in the residue field
- `check` – whether or not to check the primality of prime.

OUTPUT:

The residue field at this prime.

EXAMPLES:

```
sage: R.<x> = QQ[]
sage: K.<a> = NumberField(x^4+3*x^2-17)
sage: P = K.ideal(61).factor()[0][0]
sage: OK = K.maximal_order()
sage: OK.residue_field(P)
Residue field in abar of Fractional ideal (61, a^2 + 30)
sage: Fp.<b> = OK.residue_field(P)
sage: Fp
Residue field in b of Fractional ideal (61, a^2 + 30)
```

ring_generators()

Return generators for self as a ring.

EXAMPLES:

```
sage: K.<i> = NumberField(x^2 + 1)
sage: O = K.maximal_order(); O
Maximal Order in Number Field in i with defining polynomial x^2 + 1
sage: O.ring_generators()
[i]
```

This is an example where 2 generators are required (because 2 is an essential discriminant divisor).:

```
sage: K.<a> = NumberField(x^3 + x^2 - 2*x + 8)
sage: O = K.maximal_order(); O.basis()
[1, 1/2*a^2 + 1/2*a, a^2]
sage: O.ring_generators()
[1/2*a^2 + 1/2*a, a^2]
```

An example in a relative number field:

```
sage: K.<a, b> = NumberField([x^2 + x + 1, x^3 - 3])
sage: O = K.maximal_order()
sage: O.ring_generators()
[(-5/3*b^2 + 3*b - 2)*a - 7/3*b^2 + b + 3, (-5*b^2 - 9)*a - 5*b^2 - b, (-6*b^2 - 11)*a - 6*b
```

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

Return a primitive n-th root of unity in this order, if it contains one. If all is True, return all of them.

EXAMPLES:

```
sage: F.<alpha> = NumberField(x**2+3)
sage: F.ring_of_integers().zeta(6)
1/2*alpha + 1/2
sage: O = F.order([3*alpha])
sage: O.zeta(3)
Traceback (most recent call last):
...
ArithmeticError: There are no 3rd roots of unity in self.
```

class sage.rings.number_field.order.**RelativeOrder**(*K*, *absolute_order*, *is_maximal=None*, *check=True*)

Bases: sage.rings.number_field.order.Order

A relative order in a number field.

A relative order is an order in some relative number field

Invariants of this order may be computed with respect to the contained order.

absolute_discriminant()

Return the absolute discriminant of self, which is the discriminant of the absolute order associated to self.

OUTPUT:

an integer

EXAMPLES:

```
sage: R = EquationOrder([x^2 + 1, x^3 + 2], 'a,b')
sage: d = R.absolute_discriminant(); d
-746496
sage: d is R.absolute_discriminant()
True
sage: factor(d)
-1 * 2^10 * 3^6
```

absolute_order(*names='z'*)

Return underlying absolute order associated to this relative order.

INPUT:

- names – string (default: 'z'); name of generator of absolute extension.

Note: There *is* a default variable name, since this absolute order is frequently used for internal algorithms.

EXAMPLES:

```
sage: R = EquationOrder([x^2 + 1, x^2 - 5], 'i,g'); R
Relative Order in Number Field in i with defining polynomial x^2 + 1 over its base field
sage: R.basis()
[1, 6*i - g, -g*i + 2, 7*i - g]
```

```
sage: S = R.absolute_order(); S
Order in Number Field in z with defining polynomial x^4 - 8*x^2 + 36
sage: S.basis()
[1, 5/12*z^3 + 1/6*z, 1/2*z^2, 1/2*z^3]
```

We compute a relative order in alpha0, alpha1, then make the number field that contains the absolute order be called gamma.:

```
sage: R = EquationOrder( [x^2 + 2, x^2 - 3], 'alpha'); R
Relative Order in Number Field in alpha0 with defining polynomial x^2 + 2 over its base fiel
sage: R.absolute_order('gamma')
Order in Number Field in gamma with defining polynomial x^4 - 2*x^2 + 25
sage: R.absolute_order('gamma').basis()
[1/2*gamma^2 + 1/2, 7/10*gamma^3 + 1/10*gamma, gamma^2, gamma^3]
```

basis()

Return module basis for this relative order. This is a list of elements that generate this order over the base order.

Warning: For now this basis is actually just a basis over **Z**.

EXAMPLES:

```
sage: K.<a,b> = NumberField([x^2+1, x^2+3])
sage: O = K.order([a,b])
sage: O.basis()
[1, -2*a + b, -b*a - 2, -5*a + 3*b]
sage: z = O.1; z
-2*a + b
sage: z.absolute_minpoly()
x^4 + 14*x^2 + 1
```

index_in(*other*)

Return the index of self in other. This is a lattice index, so it is a rational number if self isn't contained in other.

INPUT:

- `other` – another order with the same ambient absolute number field.

OUTPUT:

a rational number

EXAMPLES:

```
sage: K.<a,b> = NumberField([x^3 + x + 3, x^2 + 1])
sage: R1 = K.order([3*a, 2*b])
sage: R2 = K.order([a, 4*b])
sage: R1.index_in(R2)
729/8
sage: R2.index_in(R1)
8/729
```

is_suborder(*other*)

Returns true if self is a subset of the order other.

EXAMPLES:

```
sage: K.<a,b> = NumberField([x^2 + 1, x^3 + 2])
sage: R1 = K.order([a,b])
sage: R2 = K.order([2*a,b])
sage: R3 = K.order([a + b, b + 2*a])
sage: R1.is_suborder(R2)
False
sage: R2.is_suborder(R1)
True
sage: R3.is_suborder(R1)
True
sage: R1.is_suborder(R3)
True
sage: R1 == R3
True
```

sage.rings.number_field.order.**absolute_order_from_module_generators**(*gens*, *check_integral=True*, *check_rank=True*, *check_is_ring=True*, *is_maximal=None*, *allow_subfield=False*)

INPUT:

- `gens` – list of elements of an absolute number field that generates an order in that number field as a ZZ *module*.
- `check_integral` – check that each gen is integral
- `check_rank` – check that the gens span a module of the correct rank
- `check_is_ring` – check that the module is closed under multiplication (this is very expensive)
- `is_maximal` – bool (or None); set if maximality of the generated order is known

OUTPUT:

an absolute order

EXAMPLES: We have to explicitly import the function, since it isn't meant for regular usage:

```
sage: from sage.rings.number_field.order import absolute_order_from_module_generators

sage: K.<a> = NumberField(x^4 - 5)
sage: O = K.maximal_order(); O
Maximal Order in Number Field in a with defining polynomial x^4 - 5
sage: O.basis()
[1/2*a^2 + 1/2, 1/2*a^3 + 1/2*a, a^2, a^3]
sage: O.module()
Free module of degree 4 and rank 4 over Integer Ring
Echelon basis matrix:
[1/2   0 1/2   0]
[  0 1/2   0 1/2]
[  0   0   1   0]
[  0   0   0   1]
sage: g = O.gens(); g
[1/2*a^2 + 1/2, 1/2*a^3 + 1/2*a, a^2, a^3]
sage: absolute_order_from_module_generators(g)
Order in Number Field in a with defining polynomial x^4 - 5
```

We illustrate each check flag – the output is the same but in case the function would run ever so slightly faster:

```
sage: absolute_order_from_module_generators(g,  check_is_ring=False)
Order in Number Field in a with defining polynomial x^4 - 5
sage: absolute_order_from_module_generators(g,  check_rank=False)
Order in Number Field in a with defining polynomial x^4 - 5
sage: absolute_order_from_module_generators(g,  check_integral=False)
Order in Number Field in a with defining polynomial x^4 - 5
```

Next we illustrate constructing "fake" orders to illustrate turning off various check flags:

```
sage: k.<i> = NumberField(x^2 + 1)
sage: R = absolute_order_from_module_generators([2, 2*i],  check_is_ring=False); R
Order in Number Field in i with defining polynomial x^2 + 1
sage: R.basis()
[2, 2*i]
sage: R = absolute_order_from_module_generators([k(1)],  check_rank=False); R
Order in Number Field in i with defining polynomial x^2 + 1
sage: R.basis()
[1]
```

If the order contains a non-integral element, even if we don't check that, we'll find that the rank is wrong or that the order isn't closed under multiplication:

```
sage: absolute_order_from_module_generators([1/2, i],  check_integral=False)
Traceback (most recent call last):
...
ValueError: the module span of the gens is not closed under multiplication.
sage: R = absolute_order_from_module_generators([1/2, i],  check_is_ring=False, check_integral=F
Order in Number Field in i with defining polynomial x^2 + 1
sage: R.basis()
[1/2, i]
```

We turn off all check flags and make a really messed up order:

```
sage: R = absolute_order_from_module_generators([1/2, i],  check_is_ring=False, check_integral=F
Order in Number Field in i with defining polynomial x^2 + 1
sage: R.basis()
[1/2, i]
```

An order that lives in a subfield:

```
sage: F.<alpha> = NumberField(x**4+3)
sage: F.order([alpha**2], allow_subfield=True)
Order in Number Field in alpha with defining polynomial x^4 + 3
```

sage.rings.number_field.order.**absolute_order_from_ring_generators**(*gens*, *check_is_integral=True*, *check_rank=True*, *is_maximal=None*, *allow_subfield=False*)

INPUT:

- `gens` – list of integral elements of an absolute order.
- `check_is_integral` – bool (default: True), whether to check that each generator is integral.
- `check_rank` – bool (default: True), whether to check that the ring generated by gens is of full rank.
- `is_maximal` – bool (or None); set if maximality of the generated order is known

- allow_subfield – bool (default: False), if True and the generators do not generate an order, i.e., they generate a subring of smaller rank, instead of raising an error, return an order in a smaller number field.

EXAMPLES:

```
sage: K.<a> = NumberField(x^4 - 5)
sage: K.order(a)
Order in Number Field in a with defining polynomial x^4 - 5
```

We have to explicitly import this function, since typically it is called with K.order as above.:

```
sage: from sage.rings.number_field.order import absolute_order_from_ring_generators
sage: absolute_order_from_ring_generators([a])
Order in Number Field in a with defining polynomial x^4 - 5
sage: absolute_order_from_ring_generators([3*a, 2, 6*a+1])
Order in Number Field in a with defining polynomial x^4 - 5
```

If one of the inputs is non-integral, it is an error.:

```
sage: absolute_order_from_ring_generators([a/2])
Traceback (most recent call last):
...
ValueError: each generator must be integral
```

If the gens do not generate an order, i.e., generate a ring of full rank, then it is an error.:

```
sage: absolute_order_from_ring_generators([a^2])
Traceback (most recent call last):
...
ValueError: the rank of the span of gens is wrong
```

Both checking for integrality and checking for full rank can be turned off in order to save time, though one can get nonsense as illustrated below.:

```
sage: absolute_order_from_ring_generators([a/2], check_is_integral=False)
Order in Number Field in a with defining polynomial x^4 - 5
sage: absolute_order_from_ring_generators([a^2], check_rank=False)
Order in Number Field in a with defining polynomial x^4 - 5
```

sage.rings.number_field.order.**each_is_integral**(*v*)

Return True if each element of the list v of elements of a number field is integral.

EXAMPLES:

```
sage: W.<sqrt5> = NumberField(x^2 - 5)
sage: from sage.rings.number_field.order import each_is_integral
sage: each_is_integral([sqrt5, 2, (1+sqrt5)/2])
True
sage: each_is_integral([sqrt5, (1+sqrt5)/3])
False
```

sage.rings.number_field.order.**is_NumberFieldOrder**(*R*)

Return True if R is either an order in a number field or is the ring **Z** of integers.

EXAMPLES:

```
sage: from sage.rings.number_field.order import is_NumberFieldOrder
sage: is_NumberFieldOrder(NumberField(x^2+1,'a').maximal_order())
True
sage: is_NumberFieldOrder(ZZ)
True
sage: is_NumberFieldOrder(QQ)
```

```
False
sage: is_NumberFieldOrder(45)
False
```

sage.rings.number_field.order.**relative_order_from_ring_generators**(*gens*, *check_is_integral=True*, *check_rank=True*, *is_maximal=None*, *allow_subfield=False*)

INPUT:

- `gens` – list of integral elements of an absolute order.
- `check_is_integral` – bool (default: True), whether to check that each generator is integral.
- `check_rank` – bool (default: True), whether to check that the ring generated by gens is of full rank.
- `is_maximal` – bool (or None); set if maximality of the generated order is known

EXAMPLES: We have to explicitly import this function, since it isn't meant for regular usage:

```
sage: from sage.rings.number_field.order import relative_order_from_ring_generators
sage: K.<i, a> = NumberField([x^2 + 1, x^2 - 17])
sage: R = K.base_field().maximal_order()
sage: S = relative_order_from_ring_generators([i,a]); S
Relative Order in Number Field in i with defining polynomial x^2 + 1 over its base field
```

Basis for the relative order, which is obtained by computing the algebra generated by i and a:

```
sage: S.basis()
[1, 7*i - 2*a, -a*i + 8, 25*i - 7*a]
```

CHAPTER

SEVEN

NUMBER FIELD IDEALS

AUTHORS:

- Steven Sivek (2005-05-16)
- William Stein (2007-09-06): vastly improved the doctesting
- William Stein and John Cremona (2007-01-28): new class NumberFieldFractionalIdeal now used for all except the 0 ideal
- **Radoslav Kirov and Alyson Deines (2010-06-22):** prime_to_S_part, is_S_unit, is_S_integral

We test that pickling works:

```
sage: K.<a> = NumberField(x^2 - 5)
sage: I = K.ideal(2/(5+a))
sage: I == loads(dumps(I))
True
```

class sage.rings.number_field.number_field_ideal.**LiftMap**(*OK*, *M_OK_map*, *Q*, *I*)
Class to hold data needed by lifting maps from residue fields to number field orders.

class sage.rings.number_field.number_field_ideal.**NumberFieldFractionalIdeal**(*field*, *gens*, *coerce=True*)
Bases: sage.structure.element.MultiplicativeGroupElement, sage.rings.number_field.number_field_ideal.NumberFieldIdeal

A fractional ideal in a number field.

denominator()
Return the denominator ideal of this fractional ideal. Each fractional ideal has a unique expression as N/D where N, D are coprime integral ideals; the denominator is D.

EXAMPLES:

```
sage: K.<i>=NumberField(x^2+1)
sage: I = K.ideal((3+4*i)/5); I
Fractional ideal (4/5*i + 3/5)
sage: I.denominator()
Fractional ideal (2*i + 1)
sage: I.numerator()
Fractional ideal (-i - 2)
sage: I.numerator().is_integral() and I.denominator().is_integral()
True
sage: I.numerator() + I.denominator() == K.unit_ideal()
True
```

```
sage: I.numerator()/I.denominator() == I
True
```

divides(*other*)

Returns True if this ideal divides other and False otherwise.

EXAMPLES:

```
sage: K.<a> = CyclotomicField(11); K
Cyclotomic Field of order 11 and degree 10
sage: I = K.factor(31)[0][0]; I
Fractional ideal (31, a^5 + 10*a^4 - a^3 + a^2 + 9*a - 1)
sage: I.divides(I)
True
sage: I.divides(31)
True
sage: I.divides(29)
False
```

element_1_mod(*other*)

Returns an element r in this ideal such that $1 - r$ is in other

An error is raised if either ideal is not integral of if they are not coprime.

INPUT:

- `other` – another ideal of the same field, or generators of an ideal.

OUTPUT:

An element r of the ideal self such that $1 - r$ is in the ideal other

AUTHOR: Maite Aranes (modified to use PARI's idealaddtoone by Francis Clarke)

EXAMPLES:

```
sage: K.<a> = NumberField(x^3-2)
sage: A = K.ideal(a+1); A; A.norm()
Fractional ideal (a + 1)
3
sage: B = K.ideal(a^2-4*a+2); B; B.norm()
Fractional ideal (a^2 - 4*a + 2)
68
sage: r = A.element_1_mod(B); r
-a^2 + 4*a - 1
sage: r in A
True
sage: 1-r in B
True
```

TESTS:

```
sage: K.<a> = NumberField(x^3-2)
sage: A = K.ideal(a+1)
sage: B = K.ideal(a^2-4*a+1); B; B.norm()
Fractional ideal (a^2 - 4*a + 1)
99
sage: A.element_1_mod(B)
Traceback (most recent call last):
...
TypeError: Fractional ideal (a + 1), Fractional ideal (a^2 - 4*a + 1) are not coprime ideals
```

```
sage: B = K.ideal(1/a); B
Fractional ideal (1/2*a^2)
sage: A.element_1_mod(B)
Traceback (most recent call last):
...
TypeError: Fractional ideal (1/2*a^2) is not an integral ideal
```

euler_phi()

Returns the Euler φ-function of this integral ideal.

This is the order of the multiplicative group of the quotient modulo the ideal.

An error is raised if the ideal is not integral.

EXAMPLES:

```
sage: K.<i>=NumberField(x^2+1)
sage: I = K.ideal(2+i)
sage: [r for r in I.residues() if I.is_coprime(r)]
[-2*i, -i, i, 2*i]
sage: I.euler_phi()
4
sage: J = I^3
sage: J.euler_phi()
100
sage: len([r for r in J.residues() if J.is_coprime(r)])
100
sage: J = K.ideal(3-2*i)
sage: I.is_coprime(J)
True
sage: I.euler_phi()*J.euler_phi() == (I*J).euler_phi()
True
sage: L.<b> = K.extension(x^2 - 7)
sage: L.ideal(3).euler_phi()
64
```

factor()

Factorization of this ideal in terms of prime ideals.

EXAMPLES:

```
sage: K.<a> = NumberField(x^4 + 23); K
Number Field in a with defining polynomial x^4 + 23
sage: I = K.ideal(19); I
Fractional ideal (19)
sage: F = I.factor(); F
(Fractional ideal (19, 1/2*a^2 + a - 17/2)) * (Fractional ideal (19, 1/2*a^2 - a - 17/2))
sage: type(F)
<class 'sage.structure.factorization.Factorization'>
sage: list(F)
[(Fractional ideal (19, 1/2*a^2 + a - 17/2), 1), (Fractional ideal (19, 1/2*a^2 - a - 17/2),
sage: F.prod()
Fractional ideal (19)
```

idealcoprime(*J*)

Returns l such that l*self is coprime to J.

INPUT:

- J - another integral ideal of the same field as self, which must also be integral.

OUTPUT:

- l - an element such that l*self is coprime to the ideal J

TODO: Extend the implementation to non-integral ideals.

EXAMPLES:

```
sage: k.<a> = NumberField(x^2 + 23)
sage: A = k.ideal(a+1)
sage: B = k.ideal(3)
sage: A.is_coprime(B)
False
sage: lam = A.idealcoprime(B); lam
-1/6*a + 1/6
sage: (lam*A).is_coprime(B)
True
```

ALGORITHM: Uses Pari function idealcoprime.

ideallog(*x*, *gens=None*, *check=True*)

Returns the discrete logarithm of x with respect to the generators given in the bid structure of the ideal self, or with respect to the generators gens if these are given.

INPUT:

- x - a non-zero element of the number field of self, which must have valuation equal to 0 at all prime ideals in the support of the ideal self.
- gens - a list of elements of the number field which generate $(R/I)^*$, where R is the ring of integers of the field and I is this ideal, or None. If None, use the generators calculated by idealstar().
- check - if True, do a consistency check on the results. Ignored if gens is None.

OUTPUT:

- l - a list of non-negative integers (x_i) such that $x = \prod_i g_i^{x_i}$ in $(R/I)^*$, where x_i are the generators, and the list (x_i) is lexicographically minimal with respect to this requirement. If the x_i generate independent cyclic factors of order d_i, as is the case for the default generators calculated by idealstar(), this just means that $0 \leq x_i < d_i$.

A ValueError will be raised if the elements specified in gens do not in fact generate the unit group (even if the element x is in the subgroup they generate).

EXAMPLES:

```
sage: k.<a> = NumberField(x^3 - 11)
sage: A = k.ideal(5)
sage: G = A.idealstar(2)
sage: l = A.ideallog(a^2 +3)
sage: r = G(l).value()
sage: (a^2 + 3) - r in A
True
sage: A.small_residue(r) # random
a^2 - 2
```

Examples with custom generators:

```
sage: K.<a> = NumberField(x^2 - 7)
sage: I = K.ideal(17)
sage: I.ideallog(a + 7, [1+a, 2])
[10, 3]
sage: I.ideallog(a + 7, [2, 1+a])
[0, 118]
```

```
sage: L.<b> = NumberField(x^4 - x^3 - 7*x^2 + 3*x + 2)
sage: J = L.ideal(-b^3 - b^2 - 2)
sage: u = -14*b^3 + 21*b^2 + b - 1
sage: v = 4*b^2 + 2*b - 1
sage: J.ideallog(5+2*b, [u, v], check=True)
[4, 13]
```

A non-example:

```
sage: I.ideallog(a + 7, [2])
Traceback (most recent call last):
...
ValueError: Given elements do not generate unit group -- they generate a subgroup of index 3
```

ALGORITHM: Uses Pari function `ideallog`, and (if `gens` is not None) a Hermite normal form calculation to express the result in terms of the generators `gens`.

idealstar (*flag=1*)

Returns the finite abelian group $(O_K/I)^*$, where I is the ideal self of the number field K, and O_K is the ring of integers of K.

INPUT:

- `flag` (int default 1) – when `flag` =2, it also computes the generators of the group $(O_K/I)^*$, which takes more time. By default `flag` =1 (no generators are computed). In both cases the special pari structure `bid` is computed as well. If `flag` =0 (deprecated) it computes only the group structure of $(O_K/I)^*$ (with generators) and not the special `bid` structure.

OUTPUT:

The finite abelian group $(O_K/I)^*$.

Note: Uses the pari function `idealstar`. The pari function outputs a special `bid` structure which is stored in the internal field `_bid` of the ideal (when flag=1,2). The special structure `bid` is used in the pari function `ideallog` to compute discrete logarithms.

EXAMPLES:

```
sage: k.<a> = NumberField(x^3 - 11)
sage: A = k.ideal(5)
sage: G = A.idealstar(); G
Multiplicative Abelian group isomorphic to C24 x C4
sage: G.gens()
(f0, f1)

sage: G = A.idealstar(2)
sage: G.gens()
(f0, f1)
sage: G.gens_values()    # random output
(2*a^2 - 1, 2*a^2 + 2*a - 2)
sage: all([G.gen(i).value() in k for i in range(G.ngens())])
True
```

TESTS:

```
sage: k.<a> = NumberField(x^2 + 1)
sage: k.ideal(a+1).idealstar(2)
Trivial Abelian group
```

ALGORITHM: Uses Pari function `idealstar`

invertible_residues(*reduce=True*)

Returns a iterator through a list of invertible residues modulo this integral ideal.

An error is raised if this fractional ideal is not integral.

INPUT:

- `reduce` - bool. If True (default), use `small_residue` to get small representatives of the residues.

OUTPUT:

- An iterator through a list of invertible residues modulo this ideal I, i.e. a list of elements in the ring of integers R representing the elements of $(R/I)^*$.

ALGORITHM: Use pari's `idealstar` to find the group structure and generators of the multiplicative group modulo the ideal.

EXAMPLES:

```
sage: K.<i>=NumberField(x^2+1)
sage: ires =  K.ideal(2).invertible_residues(); ires
xmrange_iter([[0, 1]], <function <lambda> at 0x...>)
sage: list(ires)
[1, -i]
sage: list(K.ideal(2+i).invertible_residues())
[1, 2, 4, 3]
sage: list(K.ideal(i).residues())
[0]
sage: list(K.ideal(i).invertible_residues())
[1]
sage: I = K.ideal(3+6*i)
sage: units=I.invertible_residues()
sage: len(list(units))==I.euler_phi()
True

sage: K.<a> = NumberField(x^3-10)
sage: I = K.ideal(a-1)
sage: len(list(I.invertible_residues())) == I.euler_phi()
True

sage: K.<z> = CyclotomicField(10)
sage: len(list(K.primes_above(3)[0].invertible_residues()))
80
```

AUTHOR: John Cremona

invertible_residues_mod(*subgp_gens=[]*, *reduce=True*)

Returns a iterator through a list of representatives for the invertible residues modulo this integral ideal, modulo the subgroup generated by the elements in the list `subgp_gens`.

INPUT:

- `subgp_gens` - either None or a list of elements of the number field of self. These need not be integral, but should be coprime to the ideal self. If the list is empty or None, the function returns an iterator through a list of representatives for the invertible residues modulo the integral ideal self.
- `reduce` - bool. If True (default), use `small_residues` to get small representatives of the residues.

Note: See also invertible_residues() for a simpler version without the subgroup.

OUTPUT:

- An iterator through a list of representatives for the invertible residues modulo self and modulo the group generated by `subgp_gens`, i.e. a list of elements in the ring of integers R representing the elements of $(R/I)^*/U$, where I is this ideal and U is the subgroup of $(R/I)^*$ generated by `subgp_gens`.

EXAMPLES:

```
sage: k.<a> = NumberField(x^2 +23)
sage: I = k.ideal(a)
sage: list(I.invertible_residues_mod([-1]))
[1, 5, 2, 10, 4, 20, 8, 17, 16, 11, 9]
sage: list(I.invertible_residues_mod([1/2]))
[1, 5]
sage: list(I.invertible_residues_mod([23]))
Traceback (most recent call last):
...
TypeError: the element must be invertible mod the ideal

sage: K.<a> = NumberField(x^3-10)
sage: I = K.ideal(a-1)
sage: len(list(I.invertible_residues_mod([]))) == I.euler_phi()
True

sage: I = K.ideal(1)
sage: list(I.invertible_residues_mod([]))
[1]

sage: K.<z> = CyclotomicField(10)
sage: len(list(K.primes_above(3)[0].invertible_residues_mod([])))
80
```

AUTHOR: Maite Aranes.

is_S_integral(S)

Return True if this fractional ideal is integral with respect to the list of primes `S`.

INPUT:

- S - a list of prime ideals (not checked if they are indeed prime).

Note: This function assumes that S is a list of prime ideals, but does not check this. This function will fail if S is not a list of prime ideals.

OUTPUT:

True, if the ideal is S-integral: that is, if the valuations of the ideal at all primes not in S are non-negative. False, otherwise.

EXAMPLES:

```
sage: K.<a> = NumberField(x^2+23)
sage: I = K.ideal(1/2)
sage: P = K.ideal(2,1/2*a - 1/2)
sage: I.is_S_integral([P])
False

sage: J = K.ideal(1/5)
sage: J.is_S_integral([K.ideal(5)])
True
```

is_S_unit(S)

Return True if this fractional ideal is a unit with respect to the list of primes `S`.

INPUT:

- S - a list of prime ideals (not checked if they are indeed prime).

Note: This function assumes that S is a list of prime ideals, but does not check this. This function will fail if S is not a list of prime ideals.

OUTPUT:

True, if the ideal is an S-unit: that is, if the valuations of the ideal at all primes not in S are zero. False, otherwise.

EXAMPLES:

```
sage: K.<a> = NumberField(x^2+23)
sage: I = K.ideal(2)
sage: P = I.factor()[0][0]
sage: I.is_S_unit([P])
False
```

is_coprime(*other*)

Returns True if this ideal is coprime to the other, else False.

INPUT:

- `other` – another ideal of the same field, or generators of an ideal.

OUTPUT:

True if self and other are coprime, else False.

Note: This function works for fractional ideals as well as integral ideals.

AUTHOR: John Cremona

EXAMPLES:

```
sage: K.<i>=NumberField(x^2+1)
sage: I = K.ideal(2+i)
sage: J = K.ideal(2-i)
sage: I.is_coprime(J)
True
sage: (I^-1).is_coprime(J^3)
True
sage: I.is_coprime(5)
False
sage: I.is_coprime(6+i)
True

# See trac \# 4536:
sage: E.<a> = NumberField(x^5 + 7*x^4 + 18*x^2 + x - 3)
sage: OE = E.ring_of_integers()
sage: i,j,k = [u[0] for u in factor(3*OE)]
sage: (i/j).is_coprime(j/k)
False
sage: (j/k).is_coprime(j/k)
False

sage: F.<a, b> = NumberField([x^2 - 2, x^2 - 3])
```

```
sage: F.ideal(3 - a*b).is_coprime(F.ideal(3))
False
```

is_maximal()

Return True if this ideal is maximal. This is equivalent to self being prime, since it is nonzero.

EXAMPLES:

```
sage: K.<a> = NumberField(x^3 + 3); K
Number Field in a with defining polynomial x^3 + 3
sage: K.ideal(5).is_maximal()
False
sage: K.ideal(7).is_maximal()
True
```

is_trivial(*proof=None*)

Returns True if this is a trivial ideal.

EXAMPLES:

```
sage: F.<a> = QuadraticField(-5)
sage: I = F.ideal(3)
sage: I.is_trivial()
False
sage: J = F.ideal(5)
sage: J.is_trivial()
False
sage: (I+J).is_trivial()
True
```

numerator()

Return the numerator ideal of this fractional ideal.

Each fractional ideal has a unique expression as N/D where N, D are coprime integral ideals. The numerator is N.

EXAMPLES:

```
sage: K.<i>=NumberField(x^2+1)
sage: I = K.ideal((3+4*i)/5); I
Fractional ideal (4/5*i + 3/5)
sage: I.denominator()
Fractional ideal (2*i + 1)
sage: I.numerator()
Fractional ideal (-i - 2)
sage: I.numerator().is_integral() and I.denominator().is_integral()
True
sage: I.numerator() + I.denominator() == K.unit_ideal()
True
sage: I.numerator()/I.denominator() == I
True
```

prime_factors()

Return a list of the prime ideal factors of self

OUTPUT: list – list of prime ideals (a new list is returned each time this function is called)

EXAMPLES:

```
sage: K.<w> = NumberField(x^2 + 23)
sage: I = ideal(w+1)
```

```
sage: I.prime_factors()
[Fractional ideal (2, 1/2*w - 1/2), Fractional ideal (2, 1/2*w + 1/2), Fractional ideal (3,
```

prime_to_S_part(*S*)

Return the part of this fractional ideal which is coprime to the prime ideals in the list `S`.

Note: This function assumes that S is a list of prime ideals, but does not check this. This function will fail if S is not a list of prime ideals.

INPUT:

- S - a list of prime ideals

OUTPUT:

A fractional ideal coprime to the primes in S, whose prime factorization is that of `self` withe the primes in S removed.

EXAMPLES:

```
sage: K.<a> = NumberField(x^2-23)
sage: I = K.ideal(24)
sage: S = [K.ideal(-a+5),K.ideal(5)]
sage: I.prime_to_S_part(S)
Fractional ideal (3)
sage: J = K.ideal(15)
sage: J.prime_to_S_part(S)
Fractional ideal (3)

sage: K.<a> = NumberField(x^5-23)
sage: I = K.ideal(24)
sage: S = [K.ideal(15161*a^4 + 28383*a^3 + 53135*a^2 + 99478*a + 186250),K.ideal(2*a^4 + 3*a
sage: I.prime_to_S_part(S)
Fractional ideal (24)
```

prime_to_idealM_part(*M*)

Version for integral ideals of the `prime_to_m_part` function over **Z**. Returns the largest divisor of self that is coprime to the ideal `M`.

INPUT:

- `M` – an integral ideal of the same field, or generators of an ideal

OUTPUT:

An ideal which is the largest divisor of self that is coprime to M.

AUTHOR: Maite Aranes

EXAMPLES:

```
sage: k.<a> = NumberField(x^2 + 23)
sage: I = k.ideal(a+1)
sage: M = k.ideal(2, 1/2*a - 1/2)
sage: J = I.prime_to_idealM_part(M); J
Fractional ideal (12, 1/2*a + 13/2)
sage: J.is_coprime(M)
True

sage: J = I.prime_to_idealM_part(2); J
Fractional ideal (3, 1/2*a + 1/2)
```

```
sage: J.is_coprime(M)
True
```

ramification_index()

Return the ramification index of this fractional ideal, assuming it is prime. Otherwise, raise a ValueError.

The ramification index is the power of this prime appearing in the factorization of the prime in **Z** that this prime lies over.

EXAMPLES:

```
sage: K.<a> = NumberField(x^2 + 2); K
Number Field in a with defining polynomial x^2 + 2
sage: f = K.factor(2); f
(Fractional ideal (a))^2
sage: f[0][0].ramification_index()
2
sage: K.ideal(13).ramification_index()
1
sage: K.ideal(17).ramification_index()
Traceback (most recent call last):
...
ValueError: Fractional ideal (17) is not a prime ideal
```

ray_class_number()

Return the order of the ray class group modulo this ideal. This is a wrapper around Pari's `bnrclassno()` function.

EXAMPLE:

```
sage: K.<z> = QuadraticField(-23)
sage: p = K.primes_above(3)[0]
sage: p.ray_class_number()
3

sage: x = polygen(K)
sage: L.<w> = K.extension(x^3 - z)
sage: I = L.ideal(5)
sage: I.ray_class_number()
5184
```

reduce(f)

Return the canonical reduction of the element of f modulo the ideal I (=self). This is an element of R (the ring of integers of the number field) that is equivalent modulo I to f.

An error is raised if this fractional ideal is not integral or the element f is not integral.

INPUT:

- `f` - an integral element of the number field

OUTPUT:

An integral element g, such that $f - g$ belongs to the ideal self and such that g is a canonical reduced representative of the coset $f + I$ (I =self) as described in the `residues` function, namely an integral element with coordinates $(r_0, \ldots, r_{n-1})$, where:

- r_i is reduced modulo d_i
- $d_i = b_i[i]$, with $b_0, b_1, \ldots, b_n$ HNF basis of the ideal self.

Note: The reduced element g is not necessarily small. To get a small g use the method `small_residue`.

EXAMPLES:

```
sage: k.<a> = NumberField(x^3 + 11)
sage: I = k.ideal(5, a^2 - a + 1)
sage: c = 4*a + 9
sage: I.reduce(c)
a^2 - 2*a
sage: c - I.reduce(c) in I
True
```

The reduced element is in the list of canonical representatives returned by the `residues` method:

```
sage: I.reduce(c) in list(I.residues())
True
```

The reduced element does not necessarily have smaller norm (use `small_residue` for that)

```
sage: c.norm()
25
sage: (I.reduce(c)).norm()
209
sage: (I.small_residue(c)).norm()
10
```

Sometimes the canonical reduced representative of 1 won't be 1 (it depends on the choice of basis for the ring of integers):

```
sage: k.<a> = NumberField(x^2 + 23)
sage: I = k.ideal(3)
sage: I.reduce(3*a + 1)
-3/2*a - 1/2
sage: k.ring_of_integers().basis()
[1/2*a + 1/2, a]
```

AUTHOR: Maite Aranes.

residue_class_degree()

Return the residue class degree of this fractional ideal, assuming it is prime. Otherwise, raise a ValueError.

The residue class degree of a prime ideal I is the degree of the extension O_K/I of its prime subfield.

EXAMPLES:

```
sage: K.<a> = NumberField(x^5 + 2); K
Number Field in a with defining polynomial x^5 + 2
sage: f = K.factor(19); f
(Fractional ideal (a^2 + a - 3)) * (Fractional ideal (-2*a^4 - a^2 + 2*a - 1)) * (Fractional
sage: [i.residue_class_degree() for i, _ in f]
[2, 2, 1]
```

residue_field(*names=None*)

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

EXAMPLES:

```
sage: K.<a> = NumberField(x^3-7)
sage: P = K.ideal(29).factor()[0][0]
sage: P.residue_field()
Residue field in abar of Fractional ideal (2*a^2 + 3*a - 10)
```

```
sage: P.residue_field('z')
Residue field in z of Fractional ideal (2*a^2 + 3*a - 10)
```

Another example:

```
sage: K.<a> = NumberField(x^3-7)
sage: P = K.ideal(389).factor()[0][0]; P
Fractional ideal (389, a^2 - 44*a - 9)
sage: P.residue_class_degree()
2
sage: P.residue_field()
Residue field in abar of Fractional ideal (389, a^2 - 44*a - 9)
sage: P.residue_field('z')
Residue field in z of Fractional ideal (389, a^2 - 44*a - 9)
sage: FF.<w> = P.residue_field()
sage: FF
Residue field in w of Fractional ideal (389, a^2 - 44*a - 9)
sage: FF((a+1)^390)
36
sage: FF(a)
w
```

An example of reduction maps to the residue field: these are defined on the whole valuation ring, i.e. the subring of the number field consisting of elements with non-negative valuation. This shows that the issue raised in trac ticket #1951 has been fixed:

```
sage: K.<i> = NumberField(x^2 + 1)
sage: P1, P2 = [g[0] for g in K.factor(5)]; (P1,P2)
(Fractional ideal (-i - 2), Fractional ideal (2*i + 1))
sage: a = 1/(1+2*i)
sage: F1, F2 = [g.residue_field() for g in [P1,P2]]; (F1,F2)
(Residue field of Fractional ideal (-i - 2), Residue field of Fractional ideal (2*i + 1))
sage: a.valuation(P1)
0
sage: F1(i/7)
4
sage: F1(a)
3
sage: a.valuation(P2)
-1
sage: F2(a)
Traceback (most recent call last):
ZeroDivisionError: Cannot reduce field element -2/5*i + 1/5 modulo Fractional ideal (2*i + 1
```

An example with a relative number field:

```
sage: L.<a,b> = NumberField([x^2 + 1, x^2 - 5])
sage: p = L.ideal((-1/2*b - 1/2)*a + 1/2*b - 1/2)
sage: R = p.residue_field(); R
Residue field in abar of Fractional ideal ((-1/2*b - 1/2)*a + 1/2*b - 1/2)
sage: R.cardinality()
9
sage: R(17)
2
sage: R((a + b)/17)
abar
sage: R(1/b)
2*abar
```

We verify that #8721 is fixed:

```
sage: L.<a, b> = NumberField([x^2 - 3, x^2 - 5])
sage: L.ideal(a).residue_field()
Residue field in abar of Fractional ideal (a)
```

residues()

Returns a iterator through a complete list of residues modulo this integral ideal.

An error is raised if this fractional ideal is not integral.

OUTPUT:

An iterator through a complete list of residues modulo the integral ideal self. This list is the set of canonical reduced representatives given by all integral elements with coordinates $(r_0, \ldots, r_{n-1})$, where:

- r_i is reduced modulo d_i
- $d_i = b_i[i]$, with $b_0, b_1, \ldots, b_n$ HNF basis of the ideal.

AUTHOR: John Cremona (modified by Maite Aranes)

EXAMPLES:

```
sage: K.<i>=NumberField(x^2+1)
sage: res =  K.ideal(2).residues(); res
xmrange_iter([[0, 1], [0, 1]], <function <lambda> at 0x...>)
sage: list(res)
[0, i, 1, i + 1]
sage: list(K.ideal(2+i).residues())
[-2*i, -i, 0, i, 2*i]
sage: list(K.ideal(i).residues())
[0]
sage: I = K.ideal(3+6*i)
sage: reps=I.residues()
sage: len(list(reps)) == I.norm()
True
sage: all([r==s or not (r-s) in I for r in reps for s in reps])  # long time (6s on sage.mat
True

sage: K.<a> = NumberField(x^3-10)
sage: I = K.ideal(a-1)
sage: len(list(I.residues())) == I.norm()
True

sage: K.<z> = CyclotomicField(11)
sage: len(list(K.primes_above(3)[0].residues())) == 3**5  # long time (5s on sage.math, 2011
True
```

small_residue(f)

Given an element f of the ambient number field, returns an element g such that $f - g$ belongs to the ideal self (which must be integral), and g is small.

Note: The reduced representative returned is not uniquely determined.

ALGORITHM: Uses Pari function `nfeltreduce`.

EXAMPLES:

```
sage: k.<a> = NumberField(x^2 + 5)
sage: I = k.ideal(a)
```

```
sage: I.small_residue(14)
4

sage: K.<a> = NumberField(x^5 + 7*x^4 + 18*x^2 + x - 3)
sage: I = K.ideal(5)
sage: I.small_residue(a^2 -13)
a^2 + 5*a - 3
```

class sage.rings.number_field.number_field_ideal.**NumberFieldIdeal**(*field*, *gens*, *coerce=True*)

Bases: sage.rings.ideal.Ideal_generic

An ideal of a number field.

S_ideal_class_log(*S*)

S-class group version of ideal_class_log().

EXAMPLES:

```
sage: K.<a> = QuadraticField(-14)
sage: S = K.primes_above(2)
sage: I = K.ideal(3, a + 1)
sage: I.S_ideal_class_log(S)
[1]
sage: I.S_ideal_class_log([])
[3]
```

TESTS:

```
sage: K.<a> = QuadraticField(-974)
sage: S = K.primes_above(2)
sage: G = K.S_class_group(S)
sage: I0 = G.0.ideal(); I1 = G.1.ideal()
sage: for p in prime_range(100):
...       for P in K.primes_above(p):
...           v = P.S_ideal_class_log(S)
...           assert(G(P) == G(I0^v[0] * I1^v[1]))
```

absolute_norm()

A synonym for norm.

EXAMPLES:

```
sage: K.<i> = NumberField(x^2 + 1)
sage: K.ideal(1 + 2*i).absolute_norm()
5
```

absolute_ramification_index()

A synonym for ramification_index.

EXAMPLES:

```
sage: K.<i> = NumberField(x^2 + 1)
sage: K.ideal(1 + i).absolute_ramification_index()
2
```

artin_symbol()

Return the Artin symbol $(K/\mathbf{Q}, P)$, where K is the number field of P =self. This is the unique element s of the decomposition group of P such that $s(x) = x^p \pmod P$ where p is the residue characteristic of P. (Here P (self) should be prime and unramified.)

See the `artin_symbol` method of the `GaloisGroup_v2` class for further documentation and examples.

EXAMPLE:

```
sage: QuadraticField(-23, 'w').primes_above(7)[0].artin_symbol()
(1,2)
```

basis()

Return an immutable sequence of elements of this ideal (note: their parent is the number field) that form a basis for this ideal viewed as a $\mathbf{Z}$ -module.

OUTPUT: basis – an immutable sequence.

EXAMPLES:

```
sage: K.<z> = CyclotomicField(7)
sage: I = K.factor(11)[0][0]
sage: I.basis()           # warning -- choice of basis can be somewhat random
[11, 11*z, 11*z^2, z^3 + 5*z^2 + 4*z + 10, z^4 + z^2 + z + 5, z^5 + z^4 + z^3 + 2*z^2 + 6*z
```

An example of a non-integral ideal.:

```
sage: J = 1/I
sage: J          # warning -- choice of generators can be somewhat random
Fractional ideal (2/11*z^5 + 2/11*z^4 + 3/11*z^3 + 2/11)
sage: J.basis()           # warning -- choice of basis can be somewhat random
[1, z, z^2, 1/11*z^3 + 7/11*z^2 + 6/11*z + 10/11, 1/11*z^4 + 1/11*z^2 + 1/11*z + 7/11, 1/11*
```

coordinates(x)

Returns the coordinate vector of x with respect to this ideal.

INPUT: `x` – an element of the number field (or ring of integers) of this ideal.

OUTPUT: List giving the coordinates of x with respect to the integral basis of the ideal. In general this will be a vector of rationals; it will consist of integers if and only if x is in the ideal.

AUTHOR: John Cremona 2008-10-31

ALGORITHM:

Uses linear algebra. Provides simpler implementations for `_contains_()`, `is_integral()` and `smallest_integer()`.

EXAMPLES:

```
sage: K.<i> = QuadraticField(-1)
sage: I = K.ideal(7+3*i)
sage: Ibasis = I.integral_basis(); Ibasis
[58, i + 41]
sage: a = 23-14*i
sage: acoords = I.coordinates(a); acoords
(597/58, -14)
sage: sum([Ibasis[j]*acoords[j] for j in range(2)]) == a
True
sage: b = 123+456*i
sage: bcoords = I.coordinates(b); bcoords
(-18573/58, 456)
sage: sum([Ibasis[j]*bcoords[j] for j in range(2)]) == b
True
sage: J = K.ideal(0)
sage: J.coordinates(0)
()
```

```
sage: J.coordinates(1)
Traceback (most recent call last):
...
TypeError: vector is not in free module
```

decomposition_group()

Return the decomposition group of self, as a subset of the automorphism group of the number field of self. Raises an error if the field isn't Galois. See the decomposition_group method of the GaloisGroup_v2 class for further examples and doctests.

EXAMPLE:

```
sage: QuadraticField(-23, 'w').primes_above(7)[0].decomposition_group()
Galois group of Number Field in w with defining polynomial x^2 + 23
```

free_module()

Return the free **Z**-module contained in the vector space associated to the ambient number field, that corresponds to this ideal.

EXAMPLES:

```
sage: K.<z> = CyclotomicField(7)
sage: I = K.factor(11)[0][0]; I
Fractional ideal (-2*z^4 - 2*z^2 - 2*z + 1)
sage: A = I.free_module()
sage: A              # warning -- choice of basis can be somewhat random
Free module of degree 6 and rank 6 over Integer Ring
User basis matrix:
[11  0  0  0  0  0]
[ 0 11  0  0  0  0]
[ 0  0 11  0  0  0]
[10  4  5  1  0  0]
[ 5  1  1  0  1  0]
[ 5  6  2  1  1  1]
```

However, the actual **Z**-module is not at all random:

```
sage: A.basis_matrix().change_ring(ZZ).echelon_form()
[ 1  0  0  5  1  1]
[ 0  1  0  1  1  7]
[ 0  0  1  7  6 10]
[ 0  0  0 11  0  0]
[ 0  0  0  0 11  0]
[ 0  0  0  0  0 11]
```

The ideal doesn't have to be integral:

```
sage: J = I^(-1)
sage: B = J.free_module()
sage: B.echelonized_basis_matrix()
[ 1/11     0     0  7/11  1/11  1/11]
[    0  1/11     0  1/11  1/11  5/11]
[    0     0  1/11  5/11  4/11 10/11]
[    0     0     0     1     0     0]
[    0     0     0     0     1     0]
[    0     0     0     0     0     1]
```

This also works for relative extensions:

```
sage: K.<a,b> = NumberField([x^2 + 1, x^2 + 2])
sage: I = K.fractional_ideal(4)
sage: I.free_module()
Free module of degree 4 and rank 4 over Integer Ring
User basis matrix:
[  4   0   0   0]
[ -3   7  -1   1]
[  3   7   1   1]
[  0 -10   0  -2]
sage: J = I^(-1); J.free_module()
Free module of degree 4 and rank 4 over Integer Ring
User basis matrix:
[  1/4     0     0     0]
[-3/16  7/16 -1/16  1/16]
[ 3/16  7/16  1/16  1/16]
[    0  -5/8     0  -1/8]
```

An example of intersecting ideals by intersecting free modules.:

```
sage: K.<a> = NumberField(x^3 + x^2 - 2*x + 8)
sage: I = K.factor(2)
sage: p1 = I[0][0]; p2 = I[1][0]
sage: N = p1.free_module().intersection(p2.free_module()); N
Free module of degree 3 and rank 3 over Integer Ring
Echelon basis matrix:
[  1 1/2 1/2]
[  0   1   1]
[  0   0   2]
sage: N.index_in(p1.free_module()).abs()
2
```

TESTS:

Sage can find the free module associated to quite large ideals quickly (see trac #4627):

```
sage: y = polygen(ZZ)
sage: M.<a> = NumberField(y^20 - 2*y^19 + 10*y^17 - 15*y^16 + 40*y^14 - 64*y^13 + 46*y^12 +
sage: M.ideal(prod(prime_range(6000, 6200))).free_module()
Free module of degree 20 and rank 20 over Integer Ring
User basis matrix:
20 x 20 dense matrix over Rational Field
```

gens_reduced(*proof=None*)

Express this ideal in terms of at most two generators, and one if possible.

This function indirectly uses `bnfisprincipal`, so set `proof=True` if you want to prove correctness (which *is* the default).

EXAMPLES:

```
sage: R.<x> = PolynomialRing(QQ)
sage: K.<a> = NumberField(x^2 + 5)
sage: K.ideal(0).gens_reduced()
(0,)
sage: J = K.ideal([a+2, 9])
sage: J.gens()
(a + 2, 9)
sage: J.gens_reduced()  # random sign
(a + 2,)
sage: K.ideal([a+2, 3]).gens_reduced()
```

```
(3, a + 2)
```

TESTS:

```
sage: len(J.gens_reduced()) == 1
True

sage: all(j.parent() is K for j in J.gens())
True
sage: all(j.parent() is K for j in J.gens_reduced())
True

sage: K.<a> = NumberField(x^4 + 10*x^2 + 20)
sage: J = K.prime_above(5)
sage: J.is_principal()
False
sage: J.gens_reduced()
(5, a)
sage: all(j.parent() is K for j in J.gens())
True
sage: all(j.parent() is K for j in J.gens_reduced())
True
```

Make sure this works with large ideals (#11836):

```
sage: R.<x> = QQ['x']
sage: L.<b> = NumberField(x^10 - 10*x^8 - 20*x^7 + 165*x^6 - 12*x^5 - 760*x^3 + 2220*x^2 + 5
sage: z_x = -96698852571685/214567261524332569 6*b^9 + 2472249905907/19506114684030233 6*b^8 +
sage: P = EllipticCurve(L, '57a1').lift_x(z_x) * 3
sage: ideal = L.fractional_ideal(P[0], P[1])
sage: ideal.is_principal(proof=False)
  ***   Warning: precision too low for generators, not given.
True
sage: len(ideal.gens_reduced(proof=False))
1
```

gens_two()

Express this ideal using exactly two generators, the first of which is a generator for the intersection of the ideal with Q.

ALGORITHM: uses PARI's `idealtwoelt` function, which runs in randomized polynomial time and is very fast in practice.

EXAMPLES:

```
sage: R.<x> = PolynomialRing(QQ)
sage: K.<a> = NumberField(x^2 + 5)
sage: J = K.ideal([a+2, 9])
sage: J.gens()
(a + 2, 9)
sage: J.gens_two()
(9, a + 2)
sage: K.ideal([a+5, a+8]).gens_two()
(3, a + 2)
sage: K.ideal(0).gens_two()
(0, 0)
```

The second generator is zero if and only if the ideal is generated by a rational, in contrast to the PARI function `idealtwoelt()`:

```
sage: I = K.ideal(12)
sage: pari(K).idealtwoelt(I)  # Note that second element is not zero
[12, [0, 12]~]
sage: I.gens_two()
(12, 0)
```

ideal_class_log(*proof=None*)

Return the output of PARI's `bnfisprincipal` for this ideal, i.e. a vector expressing the class of this ideal in terms of a set of generators for the class group.

Since it uses the PARI method `bnfisprincipal`, specify `proof=True` (this is the default setting) to prove the correctness of the output.

EXAMPLES:

When the class number is 1, the result is always the empty list:

```
sage: K.<a> = QuadraticField(-163)
sage: J = K.primes_above(random_prime(10^6))[0]
sage: J.ideal_class_log()
[]
```

An example with class group of order 2. The first ideal is not principal, the second one is:

```
sage: K.<a> = QuadraticField(-5)
sage: J = K.ideal(23).factor()[0][0]
sage: J.ideal_class_log()
[1]
sage: (J^10).ideal_class_log()
[0]
```

An example with a more complicated class group:

```
sage: K.<a, b> = NumberField([x^3 - x + 1, x^2 + 26])
sage: K.class_group()
Class group of order 18 with structure C6 x C3 of Number Field in a with defining polynomial
sage: K.primes_above(7)[0].ideal_class_log() # random
[1, 2]
```

inertia_group()

Return the inertia group of self, i.e. the set of elements s of the Galois group of the number field of self (which we assume is Galois) such that s acts trivially modulo self. This is the same as the 0th ramification group of self. See the inertia_group method of the `GaloisGroup_v2` class for further examples and doctests.

EXAMPLE:

```
sage: QuadraticField(-23, 'w').primes_above(23)[0].inertia_group()
Galois group of Number Field in w with defining polynomial x^2 + 23
```

integral_basis()

Return a list of generators for this ideal as a $\mathbf{Z}$-module.

EXAMPLES:

```
sage: R.<x> = PolynomialRing(QQ)
sage: K.<i> = NumberField(x^2 + 1)
sage: J = K.ideal(i+1)
sage: J.integral_basis()
[2, i + 1]
```

integral_split()
Return a tuple (I, d), where I is an integral ideal, and d is the smallest positive integer such that this ideal is equal to I/d.

EXAMPLES:

```
sage: R.<x> = PolynomialRing(QQ)
sage: K.<a> = NumberField(x^2-5)
sage: I = K.ideal(2/(5+a))
sage: I.is_integral()
False
sage: J,d = I.integral_split()
sage: J
Fractional ideal (-1/2*a + 5/2)
sage: J.is_integral()
True
sage: d
5
sage: I == J/d
True
```

intersection(*other*)
Return the intersection of self and other.

EXAMPLE:

```
sage: K.<a> = QuadraticField(-11)
sage: p = K.ideal((a + 1)/2); q = K.ideal((a + 3)/2)
sage: p.intersection(q) == q.intersection(p) == K.ideal(a-2)
True
```

An example with non-principal ideals:

```
sage: L.<a> = NumberField(x^3 - 7)
sage: p = L.ideal(a^2 + a + 1, 2)
sage: q = L.ideal(a+1)
sage: p.intersection(q) == L.ideal(8, 2*a + 2)
True
```

A relative example:

```
sage: L.<a,b> = NumberField([x^2 + 11, x^2 - 5])
sage: A = L.ideal([15, (-3/2*b + 7/2)*a - 8])
sage: B = L.ideal([6, (-1/2*b + 1)*a - b - 5/2])
sage: A.intersection(B) == L.ideal(-1/2*a - 3/2*b - 1)
True
```

TESTS:

Test that this works with non-integral ideals (#10767):

```
sage: K = QuadraticField(-2)
sage: I = K.ideal(1/2)
sage: I.intersection(I)
Fractional ideal (1/2)
```

is_integral()
Return True if this ideal is integral.

EXAMPLES:

```
sage: R.<x> = PolynomialRing(QQ)
sage: K.<a> = NumberField(x^5-x+1)
sage: K.ideal(a).is_integral()
True
sage: (K.ideal(1) / (3*a+1)).is_integral()
False
```

is_maximal()
Return True if this ideal is maximal. This is equivalent to self being prime and nonzero.

EXAMPLES:

```
sage: K.<a> = NumberField(x^3 + 3); K
Number Field in a with defining polynomial x^3 + 3
sage: K.ideal(5).is_maximal()
False
sage: K.ideal(7).is_maximal()
True
```

is_prime()
Return True if this ideal is prime.

EXAMPLES:

```
sage: K.<a> = NumberField(x^2 - 17); K
Number Field in a with defining polynomial x^2 - 17
sage: K.ideal(5).is_prime()   # inert prime
True
sage: K.ideal(13).is_prime()  # split
False
sage: K.ideal(17).is_prime()  # ramified
False
```

is_principal(*proof=None*)
Return True if this ideal is principal.

Since it uses the PARI method `bnfisprincipal`, specify `proof=True` (this is the default setting) to prove the correctness of the output.

EXAMPLES:

sage: K = QuadraticField(-119,'a') sage: P = K.factor(2)[1][0] sage: P.is_principal() False sage: I = P^5 sage: I.is_principal() True sage: I # random Fractional ideal (-1/2*a + 3/2) sage: P = K.ideal([2]).factor()[1][0] sage: I = P^5 sage: I.is_principal() True

is_zero()
Return True iff self is the zero ideal

Note that (0) is a `NumberFieldIdeal`, not a `NumberFieldFractionalIdeal`.

EXAMPLES:

```
sage: K.<a> = NumberField(x^2 + 2); K
Number Field in a with defining polynomial x^2 + 2
sage: K.ideal(3).is_zero()
False
sage: I=K.ideal(0); I.is_zero()
True
sage: I
Ideal (0) of Number Field in a with defining polynomial x^2 + 2
```

norm()
Return the norm of this fractional ideal as a rational number.

EXAMPLES:

```
sage: K.<a> = NumberField(x^4 + 23); K
Number Field in a with defining polynomial x^4 + 23
sage: I = K.ideal(19); I
Fractional ideal (19)
sage: factor(I.norm())
19^4
sage: F = I.factor()
sage: F[0][0].norm().factor()
19^2
```

number_field()
Return the number field that this is a fractional ideal in.

EXAMPLES:

```
sage: K.<a> = NumberField(x^2 + 2); K
Number Field in a with defining polynomial x^2 + 2
sage: K.ideal(3).number_field()
Number Field in a with defining polynomial x^2 + 2
sage: K.ideal(0).number_field() # not tested (not implemented)
Number Field in a with defining polynomial x^2 + 2
```

pari_hnf()
Return PARI's representation of this ideal in Hermite normal form.

EXAMPLES:

```
sage: R.<x> = PolynomialRing(QQ)
sage: K.<a> = NumberField(x^3 - 2)
sage: I = K.ideal(2/(5+a))
sage: I.pari_hnf()
[2, 0, 50/127; 0, 2, 244/127; 0, 0, 2/127]
```

pari_prime()
Returns a PARI prime ideal corresponding to the ideal `self`.

INPUT:

- `self` - a prime ideal.

OUTPUT: a PARI "prime ideal", i.e. a five-component vector $[p, a, e, f, b]$ representing the prime ideal $pO_K + aO_K$, e, f as usual, a as vector of components on the integral basis, b Lenstra's constant.

EXAMPLES:

```
sage: K.<i> = QuadraticField(-1)
sage: K.ideal(3).pari_prime()
[3, [3, 0]~, 1, 2, 1]
sage: K.ideal(2+i).pari_prime()
[5, [2, 1]~, 1, 1, [-2, -1; 1, -2]]
sage: K.ideal(2).pari_prime()
Traceback (most recent call last):
...
ValueError: Fractional ideal (2) is not a prime ideal
```

ramification_group(v)
Return the v'th ramification group of self, i.e. the set of elements s of the Galois group of the number field

of self (which we assume is Galois) such that s acts trivially modulo the $(v+1)$'st power of self. See the ramification_group method of the GaloisGroup class for further examples and doctests.

EXAMPLE:

```
sage: QuadraticField(-23, 'w').primes_above(23)[0].ramification_group(0)
Galois group of Number Field in w with defining polynomial x^2 + 23
sage: QuadraticField(-23, 'w').primes_above(23)[0].ramification_group(1)
Subgroup [()] of Galois group of Number Field in w with defining polynomial x^2 + 23
```

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

Return a random element of this order.

INPUT:

- *args, *kwds - Parameters passed to the random integer function. See the documentation of ZZ.random_element() for details.

OUTPUT:

A random element of this fractional ideal, computed as a random **Z**-linear combination of the basis.

EXAMPLES:

```
sage: K.<a> = NumberField(x^3 + 2)
sage: I = K.ideal(1-a)
sage: I.random_element() # random output
-a^2 - a - 19
sage: I.random_element(distribution="uniform") # random output
a^2 - 2*a - 8
sage: I.random_element(-30,30) # random output
-7*a^2 - 17*a - 75
sage: I.random_element(-100, 200).is_integral()
True
sage: I.random_element(-30,30).parent() is K
True
```

A relative example:

```
sage: K.<a, b> = NumberField([x^2 + 2, x^2 + 1000*x + 1])
sage: I = K.ideal(1-a)
sage: I.random_element() # random output
17/500002*a^3 + 737253/250001*a^2 - 1494505893/500002*a + 752473260/250001
sage: I.random_element().is_integral()
True
sage: I.random_element(-100, 200).parent() is K
True
```

reduce_equiv()

Return a small ideal that is equivalent to self in the group of fractional ideals modulo principal ideals. Very often (but not always) if self is principal then this function returns the unit ideal.

ALGORITHM: Calls pari's idealred function.

EXAMPLES:

```
sage: K.<w> = NumberField(x^2 + 23)
sage: I = ideal(w*23^5); I
Fractional ideal (6436343*w)
sage: I.reduce_equiv()
Fractional ideal (1)
sage: I = K.class_group().0.ideal()^10; I
Fractional ideal (1024, 1/2*w + 979/2)
```

```
sage: I.reduce_equiv()
Fractional ideal (2, 1/2*w - 1/2)
```

relative_norm()

A synonym for norm.

EXAMPLES:

```
sage: K.<i> = NumberField(x^2 + 1)
sage: K.ideal(1 + 2*i).relative_norm()
5
```

relative_ramification_index()

A synonym for ramification_index.

EXAMPLES:

```
sage: K.<i> = NumberField(x^2 + 1)
sage: K.ideal(1 + i).relative_ramification_index()
2
```

residue_symbol(*e*, *m*, *check=True*)

The m-th power residue symbol for an element e and the proper ideal.

$$\left(\frac{\alpha}{\mathbf{P}}\right) \equiv \alpha^{\frac{N(\mathbf{P})-1}{m}} \bmod \mathbf{P}$$

Note: accepts m=1, in which case returns 1

Note: can also be called for an element from sage.rings.number_field_element.residue_symbol

---

Note: e is coerced into the number field of self

Note: if m=2, e is an integer, and self.number_field() has absolute degree 1 (i.e. it is a copy of the rationals), then this calls kronecker_symbol, which is implemented using GMP.

INPUT:

- e - element of the number field
- m - positive integer

OUTPUT:

- an m-th root of unity in the number field

EXAMPLES:

Quadratic Residue (7 is not a square modulo 11):

```
sage: K.<a> = NumberField(x - 1)
sage: K.ideal(11).residue_symbol(7,2)
-1
```

Cubic Residue:

```
sage: K.<w> = NumberField(x^2 - x + 1)
sage: K.ideal(17).residue_symbol(w^2 + 3,3)
-w
```

The field must contain the m-th roots of unity:

```
sage: K.<w> = NumberField(x^2 - x + 1)
sage: K.ideal(17).residue_symbol(w^2 + 3,5)
Traceback (most recent call last):
...
ValueError: The residue symbol to that power is not defined for the number field
```

smallest_integer()

Return the smallest non-negative integer in $I \cap \mathbf{Z}$, where I is this ideal. If $I = 0$, returns 0.

EXAMPLES:

```
sage: R.<x> = PolynomialRing(QQ)
sage: K.<a> = NumberField(x^2+6)
sage: I = K.ideal([4,a])/7; I
Fractional ideal (2/7, 1/7*a)
sage: I.smallest_integer()
2
```

TESTS:

```
sage: K.<i> = QuadraticField(-1)
sage: P1, P2 = [P for P,e in K.factor(13)]
sage: all([(P1^i*P2^j).smallest_integer() == 13^max(i,j,0) for i in range(-3,3) for j in ran
True
sage: I = K.ideal(0)
sage: I.smallest_integer()
0

# See trac\# 4392:
sage: K.<a>=QuadraticField(-5)
sage: I=K.ideal(7)
sage: I.smallest_integer()
7

sage: K.<z> = CyclotomicField(13)
sage: a = K([-8, -4, -4, -6, 3, -4, 8, 0, 7, 4, 1, 2])
sage: I = K.ideal(a)
sage: I.smallest_integer()
146196692151
sage: I.norm()
1315770229359
sage: I.norm() / I.smallest_integer()
9
```

valuation(p)

Return the valuation of self at $\mathfrak{p}$.

INPUT:

- $\mathfrak{p}$ – a prime ideal $\mathfrak{p}$ of this number field.

OUTPUT:

(integer) The valuation of this fractional ideal at the prime $\mathfrak{p}$. If $\mathfrak{p}$ is not prime, raise a ValueError.

EXAMPLES:

```
sage: K.<a> = NumberField(x^5 + 2); K
Number Field in a with defining polynomial x^5 + 2
sage: i = K.ideal(38); i
Fractional ideal (38)
sage: i.valuation(K.factor(19)[0][0])
1
sage: i.valuation(K.factor(2)[0][0])
5
sage: i.valuation(K.factor(3)[0][0])
0
sage: i.valuation(0)
Traceback (most recent call last):
...
ValueError: p (= 0) must be nonzero
```

class sage.rings.number_field.number_field_ideal.**QuotientMap**(*K*, *M_OK_change*, *Q*, *I*)

Class to hold data needed by quotient maps from number field orders to residue fields. These are only partial maps: the exact domain is the appropriate valuation ring. For examples, see `residue_field()`.

sage.rings.number_field.number_field_ideal.**basis_to_module**(*B*, *K*)

Given a basis B of elements for a $\mathbf{Z}$-submodule of a number field K, return the corresponding $\mathbf{Z}$-submodule.

EXAMPLES:

```
sage: K.<w> = NumberField(x^4 + 1)
sage: from sage.rings.number_field.number_field_ideal import basis_to_module
sage: basis_to_module([K.0, K.0^2 + 3], K)
Free module of degree 4 and rank 2 over Integer Ring
User basis matrix:
[0 1 0 0]
[3 0 1 0]
```

sage.rings.number_field.number_field_ideal.**convert_from_idealprimedec_form**(*field*, *ideal*)

Used internally in the number field ideal implementation for converting from the form output by the PARI function `idealprimedec` to a Sage ideal.

INPUT:

- `field` - a number field
- `ideal` - a PARI prime ideal, as output by the `idealprimedec` or `idealfactor` functions

EXAMPLE:

```
sage: from sage.rings.number_field.number_field_ideal import convert_from_idealprimedec_form
sage: K.<a> = NumberField(x^2 + 3)
sage: K_bnf = gp(K.pari_bnf())
sage: ideal = K_bnf.idealprimedec(3)[1]
sage: convert_from_idealprimedec_form(K, ideal)
doctest:...: DeprecationWarning: convert_from_idealprimedec_form() is deprecated
See http://trac.sagemath.org/15767 for details.
Fractional ideal (-a)
sage: K.factor(3)
(Fractional ideal (-a))^2
```

sage.rings.number_field.number_field_ideal.**convert_to_idealprimedec_form**(*field*, *ideal*)

Used internally in the number field ideal implementation for converting to the form output by the pari function `idealprimedec` from a Sage ideal.

INPUT:

- `field` - a number field
- `ideal` - a prime ideal

NOTE:

The algorithm implemented right now is not optimal, but works. It should eventually be replaced with something better.

EXAMPLE:

```
sage: from sage.rings.number_field.number_field_ideal import convert_to_idealprimedec_form
sage: K.<a> = NumberField(x^2 + 3)
sage: P = K.ideal(a/2-3/2)
sage: convert_to_idealprimedec_form(K, P)
doctest:...: DeprecationWarning: convert_to_idealprimedec_form() is deprecated, use ideal.pari_p
See http://trac.sagemath.org/15767 for details.
[3, [1, 2]~, 2, 1, [1, 1; -1, 2]]
```

sage.rings.number_field.number_field_ideal.**is_NumberFieldFractionalIdeal**(*x*)

Return True if x is a fractional ideal of a number field.

EXAMPLES:

```
sage: from sage.rings.number_field.number_field_ideal import is_NumberFieldFractionalIdeal
sage: is_NumberFieldFractionalIdeal(2/3)
False
sage: is_NumberFieldFractionalIdeal(ideal(5))
False
sage: k.<a> = NumberField(x^2 + 2)
sage: I = k.ideal([a + 1]); I
Fractional ideal (a + 1)
sage: is_NumberFieldFractionalIdeal(I)
True
sage: Z = k.ideal(0); Z
Ideal (0) of Number Field in a with defining polynomial x^2 + 2
sage: is_NumberFieldFractionalIdeal(Z)
False
```

sage.rings.number_field.number_field_ideal.**is_NumberFieldIdeal**(*x*)

Return True if x is an ideal of a number field.

EXAMPLES:

```
sage: from sage.rings.number_field.number_field_ideal import is_NumberFieldIdeal
sage: is_NumberFieldIdeal(2/3)
False
sage: is_NumberFieldIdeal(ideal(5))
False
sage: k.<a> = NumberField(x^2 + 2)
sage: I = k.ideal([a + 1]); I
Fractional ideal (a + 1)
sage: is_NumberFieldIdeal(I)
True
sage: Z = k.ideal(0); Z
Ideal (0) of Number Field in a with defining polynomial x^2 + 2
sage: is_NumberFieldIdeal(Z)
True
```

sage.rings.number_field.number_field_ideal.**quotient_char_p**(*I*, *p*)

Given an integral ideal I that contains a prime number p, compute a vector space $V = (O_K \mod p)/(I \mod p)$, along with a homomorphism $O_K \to V$ and a section $V \to O_K$.

EXAMPLES:

```
sage: from sage.rings.number_field.number_field_ideal import quotient_char_p

sage: K.<i> = NumberField(x^2 + 1); O = K.maximal_order(); I = K.fractional_ideal(15)
sage: quotient_char_p(I, 5)[0]
Vector space quotient V/W of dimension 2 over Finite Field of size 5 where
V: Vector space of dimension 2 over Finite Field of size 5
W: Vector space of degree 2 and dimension 0 over Finite Field of size 5
Basis matrix:
[]
sage: quotient_char_p(I, 3)[0]
Vector space quotient V/W of dimension 2 over Finite Field of size 3 where
V: Vector space of dimension 2 over Finite Field of size 3
W: Vector space of degree 2 and dimension 0 over Finite Field of size 3
Basis matrix:
[]

sage: I = K.factor(13)[0][0]; I
Fractional ideal (-3*i - 2)
sage: I.residue_class_degree()
1
sage: quotient_char_p(I, 13)[0]
Vector space quotient V/W of dimension 1 over Finite Field of size 13 where
V: Vector space of dimension 2 over Finite Field of size 13
W: Vector space of degree 2 and dimension 1 over Finite Field of size 13
Basis matrix:
[1 8]
```

CHAPTER

EIGHT

RELATIVE NUMBER FIELD IDEALS

AUTHORS:

- Steven Sivek (2005-05-16)
- William Stein (2007-09-06)
- Nick Alexander (2009-01)

EXAMPLES:

```
sage: K.<a,b> = NumberField([x^2 + 1, x^2 + 2])
sage: A = K.absolute_field('z')
sage: I = A.factor(7)[0][0]
sage: from_A, to_A = A.structure()
sage: G = [from_A(z) for z in I.gens()]; G
[7, -2*b*a - 1]
sage: K.fractional_ideal(G)
Fractional ideal (2*b*a + 1)
sage: K.fractional_ideal(G).absolute_norm().factor()
7^2
```

class sage.rings.number_field.number_field_ideal_rel.**NumberFieldFractionalIdeal_rel**(*field*, *gens*, *coerce=True*)

Bases: sage.rings.number_field.number_field_ideal.NumberFieldFractionalIdeal

An ideal of a relative number field.

EXAMPLES:

```
sage: K.<a> = NumberField([x^2 + 1, x^2 + 2]); K
Number Field in a0 with defining polynomial x^2 + 1 over its base field
sage: i = K.ideal(38); i
Fractional ideal (38)

sage: K.<a0, a1> = NumberField([x^2 + 1, x^2 + 2]); K
Number Field in a0 with defining polynomial x^2 + 1 over its base field
sage: i = K.ideal([a0+1]); i # random
Fractional ideal (-a1*a0)
sage: (g, ) = i.gens_reduced(); g # random
-a1*a0
sage: (g / (a0 + 1)).is_integral()
True
sage: ((a0 + 1) / g).is_integral()
True
```

TESTS: one test fails, because ideals aren't fully integrated into the categories framework yet:

```
sage: TestSuite(i).run()
Failure in _test_category:
...
The following tests failed: _test_category
```

absolute_ideal(*names='a'*)

If this is an ideal in the extension L/K, return the ideal with the same generators in the absolute field $L/\mathbf{Q}$.

INPUT:

- names (optional) – string; name of generator of the absolute field

EXAMPLES:

```
sage: x = ZZ['x'].0
sage: K.<b> = NumberField(x^2 - 2)
sage: L.<c> = K.extension(x^2 - b)
sage: F.<m> = L.absolute_field()
```

An example of an inert ideal:

```
sage: P = F.factor(13)[0][0]; P
Fractional ideal (13)
sage: J = L.ideal(13)
sage: J.absolute_ideal()
Fractional ideal (13)
```

Now a non-trivial ideal in L that is principal in the subfield K. Since the optional 'names' argument is not passed, the generators of the absolute ideal J are returned in terms of the default field generator 'a'. This does not agree with the generator 'm' of the absolute field F defined above:

```
sage: J = L.ideal(b); J
Fractional ideal (b)
sage: J.absolute_ideal()
Fractional ideal (a^2)
sage: J.relative_norm()
Fractional ideal (2)
sage: J.absolute_norm()
4
sage: J.absolute_ideal().norm()
4
```

Now pass 'm' as the name for the generator of the absolute field:

sage: J.absolute_ideal('m') Fractional ideal (m^2)

Now an ideal not generated by an element of K:

```
sage: J = L.ideal(c); J
Fractional ideal (c)
sage: J.absolute_ideal()
Fractional ideal (a)
sage: J.absolute_norm()
2
sage: J.ideal_below()
Fractional ideal (b)
sage: J.ideal_below().norm()
2
```

absolute_norm()

Compute the absolute norm of this fractional ideal in a relative number field, returning a positive integer.

EXAMPLES:

```
sage: L.<a, b, c> = QQ.extension([x^2 - 23, x^2 - 5, x^2 - 7])
sage: I = L.ideal(a + b)
sage: I.absolute_norm()
104976
sage: I.relative_norm().relative_norm().relative_norm()
104976
```

absolute_ramification_index()

Return the absolute ramification index of this fractional ideal, assuming it is prime. Otherwise, raise a ValueError.

The absolute ramification index is the power of this prime appearing in the factorization of the rational prime that this prime lies over.

Use relative_ramification_index to obtain the power of this prime occurring in the factorization of the prime ideal of the base field that this prime lies over.

EXAMPLES:

```
sage: PQ.<X> = QQ[]
sage: F.<a, b> = NumberFieldTower([X^2 - 2, X^2 - 3])
sage: PF.<Y> = F[]
sage: K.<c> = F.extension(Y^2 - (1 + a)*(a + b)*a*b)
sage: I = K.ideal(3, c)
sage: I.absolute_ramification_index()
4
sage: I.smallest_integer()
3
sage: K.ideal(3) == I^4
True
```

element_1_mod(*other*)

Returns an element r in this ideal such that $1 - r$ is in other.

An error is raised if either ideal is not integral of if they are not coprime.

INPUT:

- `other` – another ideal of the same field, or generators of an ideal.

OUTPUT:

an element r of the ideal self such that $1 - r$ is in the ideal other.

EXAMPLES:

```
sage: K.<a, b> = NumberFieldTower([x^2 - 23, x^2 + 1])
sage: I = Ideal(2, (a - 3*b + 2)/2)
sage: J = K.ideal(a)
sage: z = I.element_1_mod(J)
sage: z in I
True
sage: 1 - z in J
True
```

factor()

Factor the ideal by factoring the corresponding ideal in the absolute number field.

EXAMPLES:

```
sage: K.<a, b> = QQ.extension([x^2 + 11, x^2 - 5])
sage: K.factor(5)
(Fractional ideal (5, (1/4*b - 1/4)*a - 1/4*b - 3/4))^2 * (Fractional ideal (5, (1/4*b - 1/4
sage: K.ideal(5).factor()
(Fractional ideal (5, (1/4*b - 1/4)*a - 1/4*b - 3/4))^2 * (Fractional ideal (5, (1/4*b - 1/4
sage: K.ideal(5).prime_factors()
[Fractional ideal (5, (1/4*b - 1/4)*a - 1/4*b - 3/4),
 Fractional ideal (5, (1/4*b - 1/4)*a - 1/4*b - 7/4)]

sage: PQ.<X> = QQ[]
sage: F.<a, b> = NumberFieldTower([X^2 - 2, X^2 - 3])
sage: PF.<Y> = F[]
sage: K.<c> = F.extension(Y^2 - (1 + a)*(a + b)*a*b)
sage: I = K.ideal(c)
sage: P = K.ideal((b*a - b - 1)*c/2 + a - 1)
sage: Q = K.ideal((b*a - b - 1)*c/2)
sage: list(I.factor()) == [(P, 2), (Q, 1)]
True
sage: I == P^2*Q
True
sage: [p.is_prime() for p in [P, Q]]
[True, True]
```

free_module()

Return this ideal as a $\mathbf{Z}$-submodule of the $\mathbf{Q}$-vector space corresponding to the ambient number field.

EXAMPLES:

```
sage: K.<a, b> = NumberField([x^3 - x + 1, x^2 + 23])
sage: I = K.ideal(a*b - 1)
sage: I.free_module()
Free module of degree 6 and rank 6 over Integer Ring
User basis matrix:
...
sage: I.free_module().is_submodule(K.maximal_order().free_module())
True
```

gens_reduced()

Return a small set of generators for this ideal. This will always return a single generator if one exists (i.e. if the ideal is principal), and otherwise two generators.

EXAMPLE:

```
sage: K.<a, b> = NumberField([x^2 + 1, x^2 - 2])
sage: I = K.ideal((a + 1)*b/2 + 1)
sage: I.gens_reduced()
(1/2*b*a + 1/2*b + 1,)
```

ideal_below()

Compute the ideal of K below this ideal of L.

EXAMPLES:

```
sage: R.<x> = QQ[]
sage: K.<a> = NumberField(x^2+6)
sage: L.<b> = K.extension(K['x'].gen()^4 + a)
sage: N = L.ideal(b)
sage: M = N.ideal_below(); M == K.ideal([-a])
True
sage: Np = L.ideal( [ L(t) for t in M.gens() ])
```

```
sage: Np.ideal_below() == M
True
sage: M.parent()
Monoid of ideals of Number Field in a with defining polynomial x^2 + 6
sage: M.ring()
Number Field in a with defining polynomial x^2 + 6
sage: M.ring() is K
True
```

This example concerns an inert ideal:

```
sage: K = NumberField(x^4 + 6*x^2 + 24, 'a')
sage: K.factor(7)
Fractional ideal (7)
sage: K0, K0_into_K, _ = K.subfields(2)[0]
sage: K0
Number Field in a0 with defining polynomial x^2 - 6*x + 24
sage: L = K.relativize(K0_into_K, 'c'); L
Number Field in c0 with defining polynomial x^2 + a0 over its base field
sage: L.base_field() is K0
True
sage: L.ideal(7)
Fractional ideal (7)
sage: L.ideal(7).ideal_below()
Fractional ideal (7)
sage: L.ideal(7).ideal_below().number_field() is K0
True
```

This example concerns an ideal that splits in the quadratic field but each factor ideal remains inert in the extension:

```
sage: len(K.factor(19))
2
sage: K0 = L.base_field(); a0 = K0.gen()
sage: len(K0.factor(19))
2
sage: w1 = -a0 + 1; P1 = K0.ideal([w1])
sage: P1.norm().factor(), P1.is_prime()
(19, True)
sage: L_into_K, K_into_L = L.structure()
sage: L.ideal(K_into_L(K0_into_K(w1))).ideal_below() == P1
True
```

The choice of embedding of quadratic field into quartic field matters:

```
sage: rho, tau = K0.embeddings(K)
sage: L1 = K.relativize(rho, 'b')
sage: L2 = K.relativize(tau, 'b')
sage: L1_into_K, K_into_L1 = L1.structure()
sage: L2_into_K, K_into_L2 = L2.structure()
sage: a = K.gen()
sage: P = K.ideal([a^2 + 5])
sage: K_into_L1(P).ideal_below() == K0.ideal([-a0 + 1])
True
sage: K_into_L2(P).ideal_below() == K0.ideal([-a0 + 5])
True
sage: K0.ideal([-a0 + 1]) == K0.ideal([-a0 + 5])
False
```

It works when the base_field is itself a relative number field:

```
sage: PQ.<X> = QQ[]
sage: F.<a, b> = NumberFieldTower([X^2 - 2, X^2 - 3])
sage: PF.<Y> = F[]
sage: K.<c> = F.extension(Y^2 - (1 + a)*(a + b)*a*b)
sage: I = K.ideal(3, c)
sage: J = I.ideal_below(); J
Fractional ideal (b)
sage: J.number_field() == F
True
```

integral_basis()

Return a basis for self as a $\mathbf{Z}$-module.

EXAMPLES:

```
sage: K.<a,b> = NumberField([x^2 + 1, x^2 - 3])
sage: I = K.ideal(17*b - 3*a)
sage: x = I.integral_basis(); x # random
[438, -b*a + 309, 219*a - 219*b, 156*a - 154*b]
```

The exact results are somewhat unpredictable, hence the `# random` flag, but we can test that they are indeed a basis:

```
sage: V, _, phi = K.absolute_vector_space()
sage: V.span([phi(u) for u in x], ZZ) == I.free_module()
True
```

integral_split()

Return a tuple (I, d), where I is an integral ideal, and d is the smallest positive integer such that this ideal is equal to I/d.

EXAMPLES:

```
sage: K.<a, b> = NumberFieldTower([x^2 - 23, x^2 + 1])
sage: I = K.ideal([a + b/3])
sage: J, d = I.integral_split()
sage: J.is_integral()
True
sage: J == d*I
True
```

is_integral()

Return True if this ideal is integral.

EXAMPLES:

```
sage: K.<a, b> = QQ.extension([x^2 + 11, x^2 - 5])
sage: I = K.ideal(7).prime_factors()[0]
sage: I.is_integral()
True
sage: (I/2).is_integral()
False
```

is_prime()

Return True if this ideal of a relative number field is prime.

EXAMPLES:

```
sage: K.<a, b> = NumberField([x^2 - 17, x^3 - 2])
sage: K.ideal(a + b).is_prime()
```

```
True
sage: K.ideal(13).is_prime()
False
```

is_principal(*proof=None*)

Return True if this ideal is principal. If so, set self.__reduced_generators, with length one.

EXAMPLES:

```
sage: K.<a, b> = NumberField([x^2 - 23, x^2 + 1])
sage: I = K.ideal([7, (-1/2*b - 3/2)*a + 3/2*b + 9/2])
sage: I.is_principal()
True
sage: I # random
Fractional ideal ((1/2*b + 1/2)*a - 3/2*b - 3/2)
```

is_zero()

Return True if this is the zero ideal.

EXAMPLE:

```
sage: K.<a, b> = NumberField([x^2 + 3, x^3 + 4])
sage: K.ideal(17).is_zero()
False
sage: K.ideal(0).is_zero()
True
```

norm()

The norm of a fractional ideal in a relative number field is deliberately unimplemented, so that a user cannot mistake the absolute norm for the relative norm, or vice versa.

EXAMPLE:

```
sage: K.<a, b> = NumberField([x^2 + 1, x^2 - 2])
sage: K.ideal(2).norm()
Traceback (most recent call last):
...
NotImplementedError: For a fractional ideal in a relative number field you must use relative
```

pari_rhnf()

Return PARI's representation of this relative ideal in Hermite normal form.

EXAMPLE:

```
sage: K.<a, b> = NumberField([x^2 + 23, x^2 - 7])
sage: I = K.ideal(2, (a + 2*b + 3)/2)
sage: I.pari_rhnf()
[[1, -2; 0, 1], [[2, 1; 0, 1], 1/2]]
```

ramification_index()

For ideals in relative number fields, `ramification_index` is deliberately not implemented in order to avoid ambiguity. Either `relative_ramification_index()` or `absolute_ramification_index()` should be used instead.

EXAMPLE:

```
sage: K.<a, b> = NumberField([x^2 + 1, x^2 - 2])
sage: K.ideal(2).ramification_index()
Traceback (most recent call last):
...
NotImplementedError: For an ideal in a relative number field you must use relative_ramificat
```

relative_norm()

Compute the relative norm of this fractional ideal in a relative number field, returning an ideal in the base field.

EXAMPLES:

```
sage: R.<x> = QQ[]
sage: K.<a> = NumberField(x^2+6)
sage: L.<b> = K.extension(K['x'].gen()^4 + a)
sage: N = L.ideal(b).relative_norm(); N
Fractional ideal (-a)
sage: N.parent()
Monoid of ideals of Number Field in a with defining polynomial x^2 + 6
sage: N.ring()
Number Field in a with defining polynomial x^2 + 6
sage: PQ.<X> = QQ[]
sage: F.<a, b> = NumberField([X^2 - 2, X^2 - 3])
sage: PF.<Y> = F[]
sage: K.<c> = F.extension(Y^2 - (1 + a)*(a + b)*a*b)
sage: K.ideal(1).relative_norm()
Fractional ideal (1)
sage: K.ideal(13).relative_norm().relative_norm()
Fractional ideal (28561)
sage: K.ideal(13).relative_norm().relative_norm().relative_norm()
815730721
sage: K.ideal(13).absolute_norm()
815730721
```

relative_ramification_index()

Return the relative ramification index of this fractional ideal, assuming it is prime. Otherwise, raise a ValueError.

The relative ramification index is the power of this prime appearing in the factorization of the prime ideal of the base field that this prime lies over.

Use absolute_ramification_index to obtain the power of this prime occurring in the factorization of the rational prime that this prime lies over.

EXAMPLES:

```
sage: PQ.<X> = QQ[]
sage: F.<a, b> = NumberFieldTower([X^2 - 2, X^2 - 3])
sage: PF.<Y> = F[]
sage: K.<c> = F.extension(Y^2 - (1 + a)*(a + b)*a*b)
sage: I = K.ideal(3, c)
sage: I.relative_ramification_index()
2
sage: I.ideal_below()  # random sign
Fractional ideal (b)
sage: I.ideal_below() == K.ideal(b)
True
sage: K.ideal(b) == I^2
True
```

residue_class_degree()

Return the residue class degree of this prime.

EXAMPLES:

```
sage: PQ.<X> = QQ[]
sage: F.<a, b> = NumberFieldTower([X^2 - 2, X^2 - 3])
```

```
sage: PF.<Y> = F[]
sage: K.<c> = F.extension(Y^2 - (1 + a)*(a + b)*a*b)
sage: [I.residue_class_degree() for I in K.ideal(c).prime_factors()]
[1, 2]
```

residues()

Returns a iterator through a complete list of residues modulo this integral ideal.

An error is raised if this fractional ideal is not integral.

EXAMPLES:

```
sage: K.<a, w> = NumberFieldTower([x^2 - 3, x^2 + x + 1])
sage: I = K.ideal(6, -w*a - w + 4)
sage: list(I.residues())[:5]
[(25/3*w - 1/3)*a + 22*w + 1,
(16/3*w - 1/3)*a + 13*w,
(7/3*w - 1/3)*a + 4*w - 1,
(-2/3*w - 1/3)*a - 5*w - 2,
(-11/3*w - 1/3)*a - 14*w - 3]
```

smallest_integer()

Return the smallest non-negative integer in $I \cap \mathbf{Z}$, where I is this ideal. If $I = 0$, returns 0.

EXAMPLES:

```
sage: K.<a, b> = NumberFieldTower([x^2 - 23, x^2 + 1])
sage: I = K.ideal([a + b])
sage: I.smallest_integer()
12
sage: [m for m in range(13) if m in I]
[0, 12]
```

valuation(p)

Return the valuation of this fractional ideal at p.

INPUT:

- p – a prime ideal $\mathfrak{p}$ of this relative number field.

OUTPUT:

(integer) The valuation of this fractional ideal at the prime $\mathfrak{p}$. If $\mathfrak{p}$ is not prime, raise a ValueError.

EXAMPLES:

```
sage: K.<a, b> = NumberField([x^2 - 17, x^3 - 2])
sage: A = K.ideal(a + b)
sage: A.is_prime()
True
sage: (A*K.ideal(3)).valuation(A)
1
sage: K.ideal(25).valuation(5)
Traceback (most recent call last):
...
ValueError: p (= Fractional ideal (5)) must be a prime
```

sage.rings.number_field.number_field_ideal_rel.**is_NumberFieldFractionalIdeal_rel**(x)

Return True if x is a fractional ideal of a relative number field.

EXAMPLES:

```
sage: from sage.rings.number_field.number_field_ideal_rel import is_NumberFieldFractionalIdeal_r
sage: from sage.rings.number_field.number_field_ideal import is_NumberFieldFractionalIdeal
sage: is_NumberFieldFractionalIdeal_rel(2/3)
False
sage: is_NumberFieldFractionalIdeal_rel(ideal(5))
False
sage: k.<a> = NumberField(x^2 + 2)
sage: I = k.ideal([a + 1]); I
Fractional ideal (a + 1)
sage: is_NumberFieldFractionalIdeal_rel(I)
False
sage: R.<x> = QQ[]
sage: K.<a> = NumberField(x^2+6)
sage: L.<b> = K.extension(K['x'].gen()^4 + a)
sage: I = L.ideal(b); I
Fractional ideal (6, b)
sage: is_NumberFieldFractionalIdeal_rel(I)
True
sage: N = I.relative_norm(); N
Fractional ideal (-a)
sage: is_NumberFieldFractionalIdeal_rel(N)
False
sage: is_NumberFieldFractionalIdeal(N)
True
```

CHAPTER

NINE

MORPHISMS BETWEEN NUMBER FIELDS

This module provides classes to represent ring homomorphisms between number fields (i.e. field embeddings).

class sage.rings.number_field.morphism.**CyclotomicFieldHomomorphism_im_gens**
Bases: sage.rings.number_field.morphism.NumberFieldHomomorphism_im_gens

class sage.rings.number_field.morphism.**CyclotomicFieldHomset**(*R*, *S*, *category=None*)
Bases: sage.rings.number_field.morphism.NumberFieldHomset

Set of homomorphisms with domain a given cyclotomic field.

EXAMPLES:

```
sage: End(CyclotomicField(16))
Automorphism group of Cyclotomic Field of order 16 and degree 8
```

list()
Return a list of all the elements of self (for which the domain is a cyclotomic field).

EXAMPLES:

```
sage: K.<z> = CyclotomicField(12)
sage: G = End(K); G
Automorphism group of Cyclotomic Field of order 12 and degree 4
sage: [g(z) for g in G]
[z, z^3 - z, -z, -z^3 + z]
sage: L.<a, b> = NumberField([x^2 + x + 1, x^4 + 1])
sage: L
Number Field in a with defining polynomial x^2 + x + 1 over its base field
sage: Hom(CyclotomicField(12), L)[3]
Ring morphism:
  From: Cyclotomic Field of order 12 and degree 4
  To:   Number Field in a with defining polynomial x^2 + x + 1 over its base field
  Defn: zeta12 |--> -b^2*a
sage: list(Hom(CyclotomicField(5), K))
[]
sage: Hom(CyclotomicField(11), L).list()
[]
```

class sage.rings.number_field.morphism.**NumberFieldHomomorphism_im_gens**
Bases: sage.rings.morphism.RingHomomorphism_im_gens

preimage(*y*)
Computes a preimage of y in the domain, provided one exists. Raises a ValueError if y has no preimage.

INPUT:

- y – an element of the codomain of self.

OUTPUT:

Returns the preimage of y in the domain, if one exists. Raises a ValueError if y has no preimage.

EXAMPLES:

```
sage: K.<a> = NumberField(x^2 - 7)
sage: L.<b> = NumberField(x^4 - 7)
sage: f = K.embeddings(L)[0]
sage: f.preimage(3*b^2 - 12/7)
3*a - 12/7
sage: f.preimage(b)
Traceback (most recent call last):
...
ValueError: Element 'b' is not in the image of this homomorphism.

sage: F.<b> = QuadraticField(23)
sage: G.<a> = F.extension(x^3+5)
sage: f = F.embeddings(G)[0]
sage: f.preimage(a^3+2*b+3)
2*b - 2
```

class sage.rings.number_field.morphism.**NumberFieldHomset**(*R*, *S*, *category=None*)

Bases: sage.rings.homset.RingHomset_generic

Set of homomorphisms with domain a given number field.

TESTS:

```
sage: H = Hom(QuadraticField(-1, 'a'), QuadraticField(-1, 'b'))
sage: TestSuite(H).run()
  Failure in _test_category:
...
The following tests failed: _test_elements
```

cardinality()

Return the order of this set of field homomorphism.

EXAMPLES:

```
sage: k.<a> = NumberField(x^2 + 1)
sage: End(k)
Automorphism group of Number Field in a with defining polynomial x^2 + 1
sage: End(k).order()
2
sage: k.<a> = NumberField(x^3 + 2)
sage: End(k).order()
1

sage: K.<a> = NumberField( [x^3 + 2, x^2 + x + 1] )
sage: End(K).order()
6
```

list()

Return a list of all the elements of self.

EXAMPLES:

```
sage: K.<a> = NumberField(x^3 - 3*x + 1)
sage: End(K).list()
[
Ring endomorphism of Number Field in a with defining polynomial x^3 - 3*x + 1
```

```
  Defn: a |--> a,
Ring endomorphism of Number Field in a with defining polynomial x^3 - 3*x + 1
  Defn: a |--> a^2 - 2,
Ring endomorphism of Number Field in a with defining polynomial x^3 - 3*x + 1
  Defn: a |--> -a^2 - a + 2
]
sage: Hom(K, CyclotomicField(9))[0] # indirect doctest
Ring morphism:
  From: Number Field in a with defining polynomial x^3 - 3*x + 1
  To:   Cyclotomic Field of order 9 and degree 6
  Defn: a |--> -zeta9^4 + zeta9^2 - zeta9
```

An example where the codomain is a relative extension:

```
sage: K.<a> = NumberField(x^3 - 2)
sage: L.<b> = K.extension(x^2 + 3)
sage: Hom(K, L).list()
[
Ring morphism:
  From: Number Field in a with defining polynomial x^3 - 2
  To:   Number Field in b with defining polynomial x^2 + 3 over its base field
  Defn: a |--> a,
Ring morphism:
  From: Number Field in a with defining polynomial x^3 - 2
  To:   Number Field in b with defining polynomial x^2 + 3 over its base field
  Defn: a |--> -1/2*a*b - 1/2*a,
Ring morphism:
  From: Number Field in a with defining polynomial x^3 - 2
  To:   Number Field in b with defining polynomial x^2 + 3 over its base field
  Defn: a |--> 1/2*a*b - 1/2*a
]
```

order()

Return the order of this set of field homomorphism.

EXAMPLES:

```
sage: k.<a> = NumberField(x^2 + 1)
sage: End(k)
Automorphism group of Number Field in a with defining polynomial x^2 + 1
sage: End(k).order()
2
sage: k.<a> = NumberField(x^3 + 2)
sage: End(k).order()
1

sage: K.<a> = NumberField( [x^3 + 2, x^2 + x + 1] )
sage: End(K).order()
6
```

class sage.rings.number_field.morphism.**RelativeNumberFieldHomomorphism_from_abs**(*parent*, *abs_hom*)

Bases: sage.rings.morphism.RingHomomorphism

A homomorphism from a relative number field to some other ring, stored as a homomorphism from the corresponding absolute field.

abs_hom()

Return the corresponding homomorphism from the absolute number field.

EXAMPLES:

```
sage: K.<a, b> = NumberField( [x^3 + 2, x^2 + x + 1] )
sage: K.hom(a, K).abs_hom()
Ring morphism:
  From: Number Field in a with defining polynomial x^6 - 3*x^5 + 6*x^4 - 3*x^3 - 9*x + 9
  To:   Number Field in a with defining polynomial x^3 + 2 over its base field
  Defn: a |--> a - b
```

im_gens()

Return the images of the generators under this map.

EXAMPLES:

```
sage: K.<a, b> = NumberField( [x^3 + 2, x^2 + x + 1] )
sage: K.hom(a, K).im_gens()
[a, b]
```

class sage.rings.number_field.morphism.**RelativeNumberFieldHomset**(*R*, *S*, *category=None*)

Bases: sage.rings.number_field.morphism.NumberFieldHomset

Set of homomorphisms with domain a given relative number field.

EXAMPLES:

We construct a homomorphism from a relative field by giving the image of a generator:

```
sage: L.<cuberoot2, zeta3> = CyclotomicField(3).extension(x^3 - 2)
sage: phi = L.hom([cuberoot2 * zeta3]); phi
Relative number field endomorphism of Number Field in cuberoot2 with defining polynomial x^3 - 2
  Defn: cuberoot2 |--> zeta3*cuberoot2
        zeta3 |--> zeta3
sage: phi(cuberoot2 + zeta3)
zeta3*cuberoot2 + zeta3
```

In fact, this phi is a generator for the Kummer Galois group of this cyclic extension:

```
sage: phi(phi(cuberoot2 + zeta3))
(-zeta3 - 1)*cuberoot2 + zeta3
sage: phi(phi(phi(cuberoot2 + zeta3)))
cuberoot2 + zeta3
```

default_base_hom()

Pick an embedding of the base field of self into the codomain of this homset. This is done in an essentially arbitrary way.

EXAMPLES:

```
sage: L.<a, b> = NumberField([x^3 - x + 1, x^2 + 23])
sage: M.<c> = NumberField(x^4 + 80*x^2 + 36)
sage: Hom(L, M).default_base_hom()
Ring morphism:
  From: Number Field in b with defining polynomial x^2 + 23
  To:   Number Field in c with defining polynomial x^4 + 80*x^2 + 36
  Defn: b |--> 1/12*c^3 + 43/6*c
```

list()

Return a list of all the elements of self (for which the domain is a relative number field).

EXAMPLES:

```
sage: K.<a, b> = NumberField([x^2 + x + 1, x^3 + 2])
sage: End(K).list()
[
Relative number field endomorphism of Number Field in a with defining polynomial x^2 + x + 1
  Defn: a |--> a
        b |--> b,
...
Relative number field endomorphism of Number Field in a with defining polynomial x^2 + x + 1
  Defn: a |--> a
        b |--> -b*a - b
]
```

An example with an absolute codomain:

```
sage: K.<a, b> = NumberField([x^2 - 3, x^2 + 2])
sage: Hom(K, CyclotomicField(24, 'z')).list()
[
Relative number field morphism:
  From: Number Field in a with defining polynomial x^2 - 3 over its base field
  To:   Cyclotomic Field of order 24 and degree 8
  Defn: a |--> z^6 - 2*z^2
        b |--> -z^5 - z^3 + z,
...
Relative number field morphism:
  From: Number Field in a with defining polynomial x^2 - 3 over its base field
  To:   Cyclotomic Field of order 24 and degree 8
  Defn: a |--> -z^6 + 2*z^2
        b |--> z^5 + z^3 - z
]
```

CHAPTER

TEN

EMBEDDINGS INTO AMBIENT FIELDS

This module provides classes to handle embeddings of number fields into ambient fields (generally $\mathbf{R}$ or $\mathbf{C}$).

class sage.rings.number_field.number_field_morphisms.**CyclotomicFieldEmbedding**
Bases: sage.rings.number_field.number_field_morphisms.NumberFieldEmbedding

Specialized class for converting cyclotomic field elements into a cyclotomic field of higher order. All the real work is done by _lift_cyclotomic_element.

class sage.rings.number_field.number_field_morphisms.**EmbeddedNumberFieldConversion**
Bases: sage.categories.map.Map

This allows one to cast one number field in another consistently, assuming they both have specified embeddings into an ambient field (by default it looks for an embedding into $\mathbf{C}$).

This is done by factoring the minimal polynomial of the input in the number field of the codomain. This may fail if the element is not actually in the given field.

ambient_field

class sage.rings.number_field.number_field_morphisms.**EmbeddedNumberFieldMorphism**
Bases: sage.rings.number_field.number_field_morphisms.NumberFieldEmbedding

This allows one to go from one number field in another consistently, assuming they both have specified embeddings into an ambient field.

If no ambient field is supplied, then the following ambient fields are tried:

- the pushout of the fields where the number fields are embedded;
- the algebraic closure of the previous pushout;
- $\mathbf{C}$.

EXAMPLES:

```
sage: K.<i> = NumberField(x^2+1,embedding=QQbar(I))
sage: L.<i> = NumberField(x^2+1,embedding=-QQbar(I))
sage: from sage.rings.number_field.number_field_morphisms import EmbeddedNumberFieldMorphism
sage: EmbeddedNumberFieldMorphism(K,L,CDF)
Generic morphism:
  From: Number Field in i with defining polynomial x^2 + 1
  To:   Number Field in i with defining polynomial x^2 + 1
  Defn: i -> -i
sage: EmbeddedNumberFieldMorphism(K,L,QQbar)
Generic morphism:
  From: Number Field in i with defining polynomial x^2 + 1
  To:   Number Field in i with defining polynomial x^2 + 1
  Defn: i -> -i
```

ambient_field

section()

EXAMPLES:

```
sage: from sage.rings.number_field.number_field_morphisms import EmbeddedNumberFieldMorphism
sage: K.<a> = NumberField(x^2-700, embedding=25)
sage: L.<b> = NumberField(x^6-700, embedding=3)
sage: f = EmbeddedNumberFieldMorphism(K, L)
sage: f(2*a-1)
2*b^3 - 1
sage: g = f.section()
sage: g(2*b^3-1)
2*a - 1
```

class sage.rings.number_field.number_field_morphisms.**NumberFieldEmbedding**

Bases: sage.categories.morphism.Morphism

If R is a lazy field, the closest root to gen_embedding will be chosen.

EXAMPLES:

```
sage: x = polygen(QQ)
sage: from sage.rings.number_field.number_field_morphisms import NumberFieldEmbedding
sage: K.<a> = NumberField(x^3-2)
sage: f = NumberFieldEmbedding(K, RLF, 1)
sage: f(a)^3
2.00000000000000?
sage: RealField(200)(f(a)^3)
2.0000000000000000000000000000000000000000000000000000000000
```

```
sage: sigma_a = K.polynomial().change_ring(CC).roots()[1][0]; sigma_a
-0.62996052494743... - 1.09112363597172*I
sage: g = NumberFieldEmbedding(K, CC, sigma_a)
sage: g(a+1)
0.37003947505256... - 1.09112363597172*I
```

gen_image()

Returns the image of the generator under this embedding.

EXAMPLES:

```
sage: f = QuadraticField(7, 'a', embedding=2).coerce_embedding()
sage: f.gen_image()
2.645751311064591?
```

sage.rings.number_field.number_field_morphisms.**closest**(*target*, *values*, *margin=1*)

This is a utility function that returns the item in values closest to target (with respect to the code{abs} function). If margin is greater than 1, and x and y are the first and second closest elements to target, then only return x if x is margin times closer to target than y, i.e. margin * abs(target-x) < abs(target-y).

TESTS:

```
sage: from sage.rings.number_field.number_field_morphisms import closest
sage: closest(1.2, [0,1,2,3,4])
1
sage: closest(1.7, [0,1,2,3,4])
2
sage: closest(1.7, [0,1,2,3,4], margin=5)
sage: closest(1.9, [0,1,2,3,4], margin=5)
2
```

```
sage: closest(.2, [-1, 1, CDF.0, -CDF.0])
1
```

sage.rings.number_field.number_field_morphisms.**create_embedding_from_approx**(*K*, *gen_image*)

This creates a morphism into from K into the parent of gen_image, choosing as the image of the generator the closest root to gen_image in its parent.

If gen_image is in a real or complex field, then it creates an image into a lazy field.

EXAMPLES:

```
sage: from sage.rings.number_field.number_field_morphisms import create_embedding_from_approx
sage: K.<a> = NumberField(x^3-x+1/10)
sage: create_embedding_from_approx(K, 1)
Generic morphism:
  From: Number Field in a with defining polynomial x^3 - x + 1/10
  To:   Real Lazy Field
  Defn: a -> 0.9456492739235915?
sage: create_embedding_from_approx(K, 0)
Generic morphism:
  From: Number Field in a with defining polynomial x^3 - x + 1/10
  To:   Real Lazy Field
  Defn: a -> 0.10103125788101081?
sage: create_embedding_from_approx(K, -1)
Generic morphism:
  From: Number Field in a with defining polynomial x^3 - x + 1/10
  To:   Real Lazy Field
  Defn: a -> -1.046680531804603?
```

We can define embeddings from one number field to another:

```
sage: L.<b> = NumberField(x^6-x^2+1/10)
sage: create_embedding_from_approx(K, b^2)
Generic morphism:
  From: Number Field in a with defining polynomial x^3 - x + 1/10
  To:   Number Field in b with defining polynomial x^6 - x^2 + 1/10
  Defn: a -> b^2
```

The if the embedding is exact, it must be valid:

```
sage: create_embedding_from_approx(K, b)
Traceback (most recent call last):
...
ValueError: b is not a root of the defining polynomial of Number Field in a with defining polync
```

sage.rings.number_field.number_field_morphisms.**matching_root**(*poly*, *target*, *ambient_field=None*, *margin=1*, *max_prec=None*)

Given a polynomial and a target, this function chooses the root that target best approximates as compared in ambient_field.

If the parent of target is exact, the equality is required, otherwise find closest root (with respect to the code{abs} function) in the ambient field to the target, and return the root of poly (if any) that approximates it best.

EXAMPLES:

```
sage: from sage.rings.number_field.number_field_morphisms import matching_root
sage: R.<x> = CC[]
sage: matching_root(x^2-2, 1.5)
```

```
1.41421356237310
sage: matching_root(x^2-2, -100.0)
-1.41421356237310
sage: matching_root(x^2-2, .00000001)
1.41421356237310
sage: matching_root(x^3-1, CDF.0)
-0.50000000000000... + 0.86602540378443...*I
sage: matching_root(x^3-x, 2, ambient_field=RR)
1.00000000000000
```

CHAPTER

ELEVEN

STRUCTURE MAPS FOR NUMBER FIELDS

Provides isomorphisms between relative and absolute presentations, to and from vector spaces, name changing maps, etc.

EXAMPLES:

```
sage: L.<cuberoot2, zeta3> = CyclotomicField(3).extension(x^3 - 2)
sage: K = L.absolute_field('a')
sage: from_K, to_K = K.structure()
sage: from_K
Isomorphism map:
  From: Number Field in a with defining polynomial x^6 - 3*x^5 + 6*x^4 - 11*x^3 + 12*x^2 + 3*x + 1
  To:   Number Field in cuberoot2 with defining polynomial x^3 - 2 over its base field
sage: to_K
Isomorphism map:
  From: Number Field in cuberoot2 with defining polynomial x^3 - 2 over its base field
  To:   Number Field in a with defining polynomial x^6 - 3*x^5 + 6*x^4 - 11*x^3 + 12*x^2 + 3*x + 1
```

class sage.rings.number_field.maps.**MapAbsoluteToRelativeNumberField**(*A*, *R*)

Bases: sage.rings.number_field.maps.NumberFieldIsomorphism

See MapRelativeToAbsoluteNumberField for examples.

class sage.rings.number_field.maps.**MapNumberFieldToVectorSpace**(*K*, *V*)

Bases: sage.categories.map.Map

A class for the isomorphism from an absolute number field to its underlying **Q**-vector space.

EXAMPLE:

```
sage: L.<a> = NumberField(x^3 - x + 1)
sage: V, fr, to = L.vector_space()
sage: type(to)
<class 'sage.rings.number_field.maps.MapNumberFieldToVectorSpace'>
```

class sage.rings.number_field.maps.**MapRelativeNumberFieldToRelativeVectorSpace**(*K*, *V*)

Bases: sage.rings.number_field.maps.NumberFieldIsomorphism

EXAMPLE:

```
sage: K.<a, b> = NumberField([x^3 - x + 1, x^2 + 23])
sage: V, fr, to = K.relative_vector_space()
sage: type(to)
<class 'sage.rings.number_field.maps.MapRelativeNumberFieldToRelativeVectorSpace'>
```

class sage.rings.number_field.maps.**MapRelativeNumberFieldToVectorSpace**(*L*, *V*, *to_K*, *to_V*)

Bases: sage.rings.number_field.maps.NumberFieldIsomorphism

The isomorphism from a relative number field to its underlying **Q**-vector space. Compare MapRelativeNumberFieldToRelativeVectorSpace.

EXAMPLES:

```
sage: K.<a> = NumberField(x^8 + 100*x^6 + x^2 + 5)
sage: L = K.relativize(K.subfields(4)[0][1], 'b'); L
Number Field in b0 with defining polynomial x^2 + a0 over its base field
sage: L_to_K, K_to_L = L.structure()

sage: V, fr, to = L.absolute_vector_space()
sage: V
Vector space of dimension 8 over Rational Field
sage: fr
Isomorphism map:
  From: Vector space of dimension 8 over Rational Field
  To:   Number Field in b0 with defining polynomial x^2 + a0 over its base field
sage: to
Isomorphism map:
  From: Number Field in b0 with defining polynomial x^2 + a0 over its base field
  To:   Vector space of dimension 8 over Rational Field
sage: type(fr), type(to)
(<class 'sage.rings.number_field.maps.MapVectorSpaceToRelativeNumberField'>,
 <class 'sage.rings.number_field.maps.MapRelativeNumberFieldToVectorSpace'>)

sage: v = V([1, 1, 1, 1, 0, 1, 1, 1])
sage: fr(v), to(fr(v)) == v
((-a0^3 + a0^2 - a0 + 1)*b0 - a0^3 - a0 + 1, True)
sage: to(L.gen()), fr(to(L.gen())) == L.gen()
((0, 1, 0, 0, 0, 0, 0, 0), True)
```

class sage.rings.number_field.maps.**MapRelativeToAbsoluteNumberField**(*R*, *A*)

Bases: sage.rings.number_field.maps.NumberFieldIsomorphism

EXAMPLES:

```
sage: K.<a> = NumberField(x^6 + 4*x^2 + 200)
sage: L = K.relativize(K.subfields(3)[0][1], 'b'); L
Number Field in b0 with defining polynomial x^2 + a0 over its base field
sage: fr, to = L.structure()
sage: fr
Relative number field morphism:
  From: Number Field in b0 with defining polynomial x^2 + a0 over its base field
  To:   Number Field in a with defining polynomial x^6 + 4*x^2 + 200
  Defn: b0 |--> a
        a0 |--> -a^2
sage: to
Ring morphism:
  From: Number Field in a with defining polynomial x^6 + 4*x^2 + 200
  To:   Number Field in b0 with defining polynomial x^2 + a0 over its base field
  Defn: a |--> b0
sage: type(fr), type(to)
(<class 'sage.rings.number_field.morphism.RelativeNumberFieldHomomorphism_from_abs'>,
 <class 'sage.rings.number_field.morphism.NumberFieldHomomorphism_im_gens'>)

sage: M.<c> = L.absolute_field(); M
```

```
Number Field in c with defining polynomial x^6 + 4*x^2 + 200
sage: fr, to = M.structure()
sage: fr
Isomorphism map:
  From: Number Field in c with defining polynomial x^6 + 4*x^2 + 200
  To:   Number Field in b0 with defining polynomial x^2 + a0 over its base field
sage: to
Isomorphism map:
  From: Number Field in b0 with defining polynomial x^2 + a0 over its base field
  To:   Number Field in c with defining polynomial x^6 + 4*x^2 + 200
sage: type(fr), type(to)
(<class 'sage.rings.number_field.maps.MapAbsoluteToRelativeNumberField'>,
 <class 'sage.rings.number_field.maps.MapRelativeToAbsoluteNumberField'>)
sage: fr(M.gen()), to(fr(M.gen())) == M.gen()
(b0, True)
sage: to(L.gen()), fr(to(L.gen())) == L.gen()
(c, True)
sage: (to * fr)(M.gen()) == M.gen(), (fr * to)(L.gen()) == L.gen()
(True, True)
```

class sage.rings.number_field.maps.**MapRelativeVectorSpaceToRelativeNumberField**(*V*, *K*)

Bases: sage.rings.number_field.maps.NumberFieldIsomorphism

EXAMPLES:

```
sage: L.<b> = NumberField(x^4 + 3*x^2 + 1)
sage: K = L.relativize(L.subfields(2)[0][1], 'a'); K
Number Field in a0 with defining polynomial x^2 - b0*x + 1 over its base field
sage: V, fr, to = K.relative_vector_space()
sage: V
Vector space of dimension 2 over Number Field in b0 with defining polynomial x^2 + 1
sage: fr
Isomorphism map:
  From: Vector space of dimension 2 over Number Field in b0 with defining polynomial x^2 + 1
  To:   Number Field in a0 with defining polynomial x^2 - b0*x + 1 over its base field
sage: type(fr)
<class 'sage.rings.number_field.maps.MapRelativeVectorSpaceToRelativeNumberField'>

sage: a0 = K.gen(); b0 = K.base_field().gen()
sage: fr(to(a0 + 2*b0)), fr(V([0, 1])), fr(V([b0, 2*b0]))
(a0 + 2*b0, a0, 2*b0*a0 + b0)
sage: (fr * to)(K.gen()) == K.gen()
True
sage: (to * fr)(V([1, 2])) == V([1, 2])
True
```

class sage.rings.number_field.maps.**MapVectorSpaceToNumberField**(*V*, *K*)

Bases: sage.rings.number_field.maps.NumberFieldIsomorphism

The map to an absolute number field from its underlying $\mathbf{Q}$-vector space.

EXAMPLES:

```
sage: K.<a> = NumberField(x^4 + 3*x + 1)
sage: V, fr, to = K.vector_space()
sage: V
Vector space of dimension 4 over Rational Field
sage: fr
Isomorphism map:
```

```
  From: Vector space of dimension 4 over Rational Field
  To:   Number Field in a with defining polynomial x^4 + 3*x + 1
sage: to
Isomorphism map:
  From: Number Field in a with defining polynomial x^4 + 3*x + 1
  To:   Vector space of dimension 4 over Rational Field
sage: type(fr), type(to)
(<class 'sage.rings.number_field.maps.MapVectorSpaceToNumberField'>,
 <class 'sage.rings.number_field.maps.MapNumberFieldToVectorSpace'>)

sage: fr.is_injective(), fr.is_surjective()
(True, True)

sage: fr.domain(), to.codomain()
(Vector space of dimension 4 over Rational Field, Vector space of dimension 4 over Rational Fiel
sage: to.domain(), fr.codomain()
(Number Field in a with defining polynomial x^4 + 3*x + 1, Number Field in a with defining polyn
sage: fr * to
Composite map:
  From: Number Field in a with defining polynomial x^4 + 3*x + 1
  To:   Number Field in a with defining polynomial x^4 + 3*x + 1
  Defn:   Isomorphism map:
          From: Number Field in a with defining polynomial x^4 + 3*x + 1
          To:   Vector space of dimension 4 over Rational Field
        then
          Isomorphism map:
          From: Vector space of dimension 4 over Rational Field
          To:   Number Field in a with defining polynomial x^4 + 3*x + 1
sage: to * fr
Composite map:
  From: Vector space of dimension 4 over Rational Field
  To:   Vector space of dimension 4 over Rational Field
  Defn:   Isomorphism map:
          From: Vector space of dimension 4 over Rational Field
          To:   Number Field in a with defining polynomial x^4 + 3*x + 1
        then
          Isomorphism map:
          From: Number Field in a with defining polynomial x^4 + 3*x + 1
          To:   Vector space of dimension 4 over Rational Field

sage: to(a), to(a + 1)
((0, 1, 0, 0), (1, 1, 0, 0))
sage: fr(to(a)), fr(V([0, 1, 2, 3]))
(a, 3*a^3 + 2*a^2 + a)
```

class sage.rings.number_field.maps.**MapVectorSpaceToRelativeNumberField**(*V*, *L*, *from_V*, *from_K*)

Bases: sage.rings.number_field.maps.NumberFieldIsomorphism

The isomorphism to a relative number field from its underlying **Q**-vector space. Compare MapRelativeVectorSpaceToRelativeNumberField.

EXAMPLES:

```
sage: L.<a, b> = NumberField([x^2 + 3, x^2 + 5])
sage: V, fr, to = L.absolute_vector_space()
sage: type(fr)
<class 'sage.rings.number_field.maps.MapVectorSpaceToRelativeNumberField'>
```

class sage.rings.number_field.maps.**NameChangeMap**(*K*, *L*)

Bases: sage.rings.number_field.maps.NumberFieldIsomorphism

A map between two isomorphic number fields with the same defining polynomial but different variable names.

EXAMPLE:

```
sage: K.<a> = NumberField(x^2 - 3)
sage: L.<b> = K.change_names()
sage: from_L, to_L = L.structure()
sage: from_L
Isomorphism given by variable name change map:
  From: Number Field in b with defining polynomial x^2 - 3
  To:   Number Field in a with defining polynomial x^2 - 3
sage: to_L
Isomorphism given by variable name change map:
  From: Number Field in a with defining polynomial x^2 - 3
  To:   Number Field in b with defining polynomial x^2 - 3
sage: type(from_L), type(to_L)
(<class 'sage.rings.number_field.maps.NameChangeMap'>, <class 'sage.rings.number_field.maps.Name
```

class sage.rings.number_field.maps.**NumberFieldIsomorphism**

Bases: sage.categories.map.Map

A base class for various isomorphisms between number fields and vector spaces.

EXAMPLE:

```
sage: K.<a> = NumberField(x^4 + 3*x + 1)
sage: V, fr, to = K.vector_space()
sage: isinstance(fr, sage.rings.number_field.maps.NumberFieldIsomorphism)
True
```

is_injective()

EXAMPLE:

```
sage: K.<a> = NumberField(x^4 + 3*x + 1)
sage: V, fr, to = K.vector_space()
sage: fr.is_injective()
True
```

is_surjective()

EXAMPLE:

```
sage: K.<a> = NumberField(x^4 + 3*x + 1)
sage: V, fr, to = K.vector_space()
sage: fr.is_surjective()
True
```

CHAPTER

TWELVE

CLASS GROUPS OF NUMBER FIELDS

An element of a class group is stored as a pair consisting of both an explicit ideal in that ideal class, and a list of exponents giving that ideal class in terms of the generators of the parent class group. These can be accessed with the `ideal()` and `exponents()` methods respectively.

EXAMPLES:

```
sage: K.<a> = NumberField(x^2 + 23)
sage: I = K.class_group().gen(); I
Fractional ideal class (2, 1/2*a - 1/2)
sage: I.ideal()
Fractional ideal (2, 1/2*a - 1/2)
sage: I.exponents()
(1,)

sage: I.ideal() * I.ideal()
Fractional ideal (4, 1/2*a + 3/2)
sage: (I.ideal() * I.ideal()).reduce_equiv()
Fractional ideal (2, 1/2*a + 1/2)
sage: J = I * I; J    # class group multiplication is automatically reduced
Fractional ideal class (2, 1/2*a + 1/2)
sage: J.ideal()
Fractional ideal (2, 1/2*a + 1/2)
sage: J.exponents()
(2,)

sage: I * I.ideal()   # ideal classes coerce to their representative ideal
Fractional ideal (4, 1/2*a + 3/2)

sage: O = K.OK(); O
Maximal Order in Number Field in a with defining polynomial x^2 + 23
sage: O*(2, 1/2*a + 1/2)
Fractional ideal (2, 1/2*a + 1/2)
sage: (O*(2, 1/2*a + 1/2)).is_principal()
False
sage: (O*(2, 1/2*a + 1/2))^3
Fractional ideal (1/2*a - 3/2)
```

class `sage.rings.number_field.class_group.`**`ClassGroup`**(*gens_orders*, *names*, *number_field*, *gens*, *proof=True*)

Bases: `sage.groups.abelian_gps.values.AbelianGroupWithValues_class`

The class group of a number field.

EXAMPLES:

```
sage: K.<a> = NumberField(x^2 + 23)
sage: G = K.class_group(); G
Class group of order 3 with structure C3 of Number Field in a with defining polynomial x^2 + 23
sage: G.category()
Category of finite commutative groups
```

Note the distinction between abstract generators, their ideal, and exponents:

```
sage: C = NumberField(x^2 + 120071, 'a').class_group(); C
Class group of order 500 with structure C250 x C2
of Number Field in a with defining polynomial x^2 + 120071
sage: c = C.gen(0)
sage: c  # random
Fractional ideal class (5, 1/2*a + 3/2)
sage: c.ideal()  # random
Fractional ideal (5, 1/2*a + 3/2)
sage: c.ideal() is c.value()   # alias
True
sage: c.exponents()
(1, 0)
```

Element
alias of FractionalIdealClass

gens_ideals()
Return generating ideals for the (S-)class group.

This is an alias for gens_values().

OUTPUT:

A tuple of ideals, one for each abstract Abelian group generator.

EXAMPLES:

```
sage: K.<a> = NumberField(x^4 + 23)
sage: K.class_group().gens_ideals()   # random gens (platform dependent)
(Fractional ideal (2, 1/4*a^3 - 1/4*a^2 + 1/4*a - 1/4),)

sage: C = NumberField(x^2 + x + 23899, 'a').class_group(); C
Class group of order 68 with structure C34 x C2 of Number Field
in a with defining polynomial x^2 + x + 23899
sage: C.gens()
(Fractional ideal class (7, a + 5), Fractional ideal class (5, a + 3))
sage: C.gens_ideals()
(Fractional ideal (7, a + 5), Fractional ideal (5, a + 3))
```

number_field()
Return the number field that this (S-)class group is attached to.

EXAMPLES:

```
sage: C = NumberField(x^2 + 23, 'w').class_group(); C
Class group of order 3 with structure C3 of Number Field in w with defining polynomial x^2 +
sage: C.number_field()
Number Field in w with defining polynomial x^2 + 23

sage: K.<a> = QuadraticField(-14)
sage: CS = K.S_class_group(K.primes_above(2))
sage: CS.number_field()
Number Field in a with defining polynomial x^2 + 14
```

class sage.rings.number_field.class_group.**FractionalIdealClass**(*parent*, *element*, *ideal=None*)

Bases: sage.groups.abelian_gps.values.AbelianGroupWithValuesElement

A fractional ideal class in a number field.

EXAMPLES:

```
sage: G = NumberField(x^2 + 23,'a').class_group(); G
Class group of order 3 with structure C3 of Number Field in a with defining polynomial x^2 + 23
sage: I = G.0; I
Fractional ideal class (2, 1/2*a - 1/2)
sage: I.ideal()
Fractional ideal (2, 1/2*a - 1/2)
```

EXAMPLES::

```
sage: K.<w>=QuadraticField(-23)
sage: OK=K.ring_of_integers()
sage: C=OK.class_group()
sage: P2a,P2b=[P for P,e in (2*OK).factor()]
sage: c = C(P2a); c
Fractional ideal class (2, 1/2*w - 1/2)
sage: c.gens()
(2, 1/2*w - 1/2)
```

gens()

Return generators for a representative ideal in this (S-)ideal class.

EXAMPLES:

```
sage: K.<w>=QuadraticField(-23)
sage: OK = K.ring_of_integers()
sage: C = OK.class_group()
sage: P2a,P2b=[P for P,e in (2*OK).factor()]
sage: c = C(P2a); c
Fractional ideal class (2, 1/2*w - 1/2)
sage: c.gens()
(2, 1/2*w - 1/2)
```

ideal()

Return a representative ideal in this ideal class.

EXAMPLE:

```
sage: K.<w>=QuadraticField(-23)
sage: OK=K.ring_of_integers()
sage: C=OK.class_group()
sage: P2a,P2b=[P for P,e in (2*OK).factor()]
sage: c=C(P2a); c
Fractional ideal class (2, 1/2*w - 1/2)
sage: c.ideal()
Fractional ideal (2, 1/2*w - 1/2)
```

inverse()

Return the multiplicative inverse of this ideal class.

EXAMPLE:

```
sage: K.<a> = NumberField(x^3 - 3*x + 8); G = K.class_group()
sage: G(2, a).inverse()
Fractional ideal class (2, a^2 + 2*a - 1)
sage: ~G(2, a)
Fractional ideal class (2, a^2 + 2*a - 1)
```

is_principal()

Returns True iff this ideal class is the trivial (principal) class

EXAMPLES:

```
sage: K.<w>=QuadraticField(-23)
sage: OK=K.ring_of_integers()
sage: C=OK.class_group()
sage: P2a,P2b=[P for P,e in (2*OK).factor()]
sage: c=C(P2a)
sage: c.is_principal()
False
sage: (c^2).is_principal()
False
sage: (c^3).is_principal()
True
```

reduce()

Return representative for this ideal class that has been reduced using PARI's idealred.

EXAMPLES:

```
sage: k.<a> = NumberField(x^2 + 20072); G = k.class_group(); G
Class group of order 76 with structure C38 x C2
of Number Field in a with defining polynomial x^2 + 20072
sage: I = (G.0)^11; I
Fractional ideal class (41, 1/2*a + 5)
sage: J = G(I.ideal()^5); J
Fractional ideal class (115856201, 1/2*a + 40407883)
sage: J.reduce()
Fractional ideal class (57, 1/2*a + 44)
sage: J == I^5
True
```

representative_prime(*norm_bound=1000*)

Return a prime ideal in this ideal class.

INPUT:

`norm_bound` (positive integer) – upper bound on the norm of primes tested.

EXAMPLE:

```
sage: K.<a> = NumberField(x^2+31)
sage: K.class_number()
3
sage: Cl = K.class_group()
sage: [c.representative_prime() for c in Cl]
[Fractional ideal (3),
Fractional ideal (2, 1/2*a + 1/2),
Fractional ideal (2, 1/2*a - 1/2)]

sage: K.<a> = NumberField(x^2+223)
sage: K.class_number()
7
```

```
sage: Cl = K.class_group()
sage: [c.representative_prime() for c in Cl]
[Fractional ideal (3),
Fractional ideal (2, 1/2*a + 1/2),
Fractional ideal (17, 1/2*a + 7/2),
Fractional ideal (7, 1/2*a - 1/2),
Fractional ideal (7, 1/2*a + 1/2),
Fractional ideal (17, 1/2*a + 27/2),
Fractional ideal (2, 1/2*a - 1/2)]
```

class sage.rings.number_field.class_group.**SClassGroup**(*gens_orders*, *names*, *number_field*, *gens*, *S*, *proof=True*)

Bases: sage.rings.number_field.class_group.ClassGroup

The S-class group of a number field.

EXAMPLES:

```
sage: K.<a> = QuadraticField(-14)
sage: S = K.primes_above(2)
sage: K.S_class_group(S).gens()   # random gens (platform dependent)
(Fractional S-ideal class (3, a + 2),)

sage: K.<a> = QuadraticField(-974)
sage: CS = K.S_class_group(K.primes_above(2)); CS
S-class group of order 18 with structure C6 x C3
of Number Field in a with defining polynomial x^2 + 974
sage: CS.gen(0) # random
Fractional S-ideal class (3, a + 2)
sage: CS.gen(1) # random
Fractional S-ideal class (31, a + 24)
```

Element

alias of SFractionalIdealClass

S()

Return the set (or rather tuple) of primes used to define this class group.

EXAMPLES:

```
sage: K.<a> = QuadraticField(-14)
sage: I = K.ideal(2,a)
sage: S = (I,)
sage: CS = K.S_class_group(S);CS
S-class group of order 2 with structure C2 of Number Field in a with defining polynomial x^2
sage: T = tuple([])
sage: CT = K.S_class_group(T);CT
S-class group of order 4 with structure C4 of Number Field in a with defining polynomial x^2
sage: CS.S()
(Fractional ideal (2, a),)
sage: CT.S()
()
```

class sage.rings.number_field.class_group.**SFractionalIdealClass**(*parent*, *element*, *ideal=None*)

Bases: sage.rings.number_field.class_group.FractionalIdealClass

An S-fractional ideal class in a number field for a tuple of primes S.

EXAMPLES:

```
sage: K.<a> = QuadraticField(-14)
sage: I = K.ideal(2,a)
sage: S = (I,)
sage: CS = K.S_class_group(S)
sage: J = K.ideal(7,a)
sage: G = K.ideal(3,a+1)
sage: CS(I)
Trivial S-ideal class
sage: CS(J)
Trivial S-ideal class
sage: CS(G)
Fractional S-ideal class (3, a + 1)
```

EXAMPLES:

```
sage: K.<a> = QuadraticField(-14)
sage: I = K.ideal(2,a)
sage: S = (I,)
sage: CS = K.S_class_group(S)
sage: J = K.ideal(7,a)
sage: G = K.ideal(3,a+1)
sage: CS(I).ideal()
Fractional ideal (2, a)
sage: CS(J).ideal()
Fractional ideal (7, a)
sage: CS(G).ideal()
Fractional ideal (3, a + 1)
```

EXAMPLES:

```
sage: K.<a> = QuadraticField(-14)
sage: I = K.ideal(2,a)
sage: S = (I,)
sage: CS = K.S_class_group(S)
sage: G = K.ideal(3,a+1)
sage: CS(G).inverse()
Fractional S-ideal class (3, a + 2)
```

TESTS:

```
sage: K.<a> = QuadraticField(-14)
sage: I = K.ideal(2,a)
sage: S = (I,)
sage: CS = K.S_class_group(S)
sage: J = K.ideal(7,a)
sage: G = K.ideal(3,a+1)
sage: CS(I).order()
1
sage: CS(J).order()
1
sage: CS(G).order()
2
```

CHAPTER

THIRTEEN

GALOIS GROUPS OF NUMBER FIELDS

AUTHORS:

- William Stein (2004, 2005): initial version
- David Loeffler (2009): rewrite to give explicit homomorphism groups

TESTS:

Standard test of pickleability:

```
sage: G = NumberField(x^3 + 2, 'alpha').galois_group(type="pari"); G
Galois group PARI group [6, -1, 2, "S3"] of degree 3 of the Number Field in alpha with defining polyn
sage: G == loads(dumps(G))
True

sage: G = NumberField(x^3 + 2, 'alpha').galois_group(names='beta'); G
Galois group of Galois closure in beta of Number Field in alpha with defining polynomial x^3 + 2
sage: G == loads(dumps(G))
True
```

sage.rings.number_field.galois_group.**GaloisGroup**
alias of GaloisGroup_v1

class sage.rings.number_field.galois_group.**GaloisGroupElement**
Bases: sage.groups.perm_gps.permgroup_element.PermutationGroupElement

An element of a Galois group. This is stored as a permutation, but may also be made to act on elements of the field (generally returning elements of its Galois closure).

EXAMPLE:

```
sage: K.<w> = QuadraticField(-7); G = K.galois_group()
sage: G[1]
(1,2)
sage: G[1](w + 2)
-w + 2

sage: L.<v> = NumberField(x^3 - 2); G = L.galois_group(names='y')
sage: G[4]
(1,5)(2,4)(3,6)
sage: G[4](v)
1/18*y^4
sage: G[4](G[4](v))
-1/36*y^4 - 1/2*y
sage: G[4](G[4](G[4](v)))
1/18*y^4
```

as_hom()
Return the homomorphism L -> L corresponding to self, where L is the Galois closure of the ambient number field.

EXAMPLE:

```
sage: G = QuadraticField(-7,'w').galois_group()
sage: G[1].as_hom()
Ring endomorphism of Number Field in w with defining polynomial x^2 + 7
Defn: w |--> -w
```

ramification_degree(*P*)
Return the greatest value of v such that s acts trivially modulo P^v. Should only be used if P is prime and s is in the decomposition group of P.

EXAMPLE:

```
sage: K.<b> = NumberField(x^3 - 3, 'a').galois_closure()
sage: G = K.galois_group()
sage: P = K.primes_above(3)[0]
sage: s = hom(K, K, 1/18*b^4 - 1/2*b)
sage: G(s).ramification_degree(P)
4
```

class sage.rings.number_field.galois_group.**GaloisGroup_subgroup**(*ambient*, *elts*)
Bases: sage.rings.number_field.galois_group.GaloisGroup_v2

A subgroup of a Galois group, as returned by functions such as decomposition_group.

fixed_field()
Return the fixed field of this subgroup (as a subfield of the Galois closure of the number field associated to the ambient Galois group).

EXAMPLE:

```
sage: L.<a> = NumberField(x^4 + 1)
sage: G = L.galois_group()
sage: H = G.decomposition_group(L.primes_above(3)[0])
sage: H.fixed_field()
(Number Field in a0 with defining polynomial x^2 + 2, Ring morphism:
From: Number Field in a0 with defining polynomial x^2 + 2
To:   Number Field in a with defining polynomial x^4 + 1
Defn: a0 |--> a^3 + a)
```

class sage.rings.number_field.galois_group.**GaloisGroup_v1**(*group*, *number_field*)
Bases: sage.structure.sage_object.SageObject

A wrapper around a class representing an abstract transitive group.

This is just a fairly minimal object at present. To get the underlying group, do G.group(), and to get the corresponding number field do G.number_field(). For a more sophisticated interface use the type=None option.

EXAMPLES:

```
sage: K = QQ[2^(1/3)]
sage: G = K.galois_group(type="pari"); G
Galois group PARI group [6, -1, 2, "S3"] of degree 3 of the Number Field in a with defining poly
sage: G.order()
6
sage: G.group()
PARI group [6, -1, 2, "S3"] of degree 3
```

```
sage: G.number_field()
Number Field in a with defining polynomial x^3 - 2
```

group()
Return the underlying abstract group.

EXAMPLES:

```
sage: G = NumberField(x^3 + 2*x + 2, 'theta').galois_group(type="pari")
sage: H = G.group(); H
PARI group [6, -1, 2, "S3"] of degree 3
sage: P = H.permutation_group(); P  # optional - database_gap
Transitive group number 2 of degree 3
sage: list(P)                       # optional - database_gap
[(), (1,2), (1,2,3), (2,3), (1,3,2), (1,3)]
```

number_field()
Return the number field of which this is the Galois group.

EXAMPLES:

```
sage: G = NumberField(x^6 + 2, 't').galois_group(type="pari"); G
Galois group PARI group [12, -1, 3, "D(6) = S(3)[x]2"] of degree 6 of the Number Field in t
sage: G.number_field()
Number Field in t with defining polynomial x^6 + 2
```

order()
Return the order of this Galois group.

EXAMPLES:

```
sage: G = NumberField(x^5 + 2, 'theta_1').galois_group(type="pari"); G
Galois group PARI group [20, -1, 3, "F(5) = 5:4"] of degree 5 of the Number Field in theta_1
sage: G.order()
20
```

class sage.rings.number_field.galois_group.**GaloisGroup_v2**(*number_field*, *names=None*)
Bases: sage.groups.perm_gps.permgroup.PermutationGroup_generic

The Galois group of an (absolute) number field.

Note: We define the Galois group of a non-normal field K to be the Galois group of its Galois closure L, and elements are stored as permutations of the roots of the defining polynomial of L, *not* as permutations of the roots (in L) of the defining polynomial of K. The latter would probably be preferable, but is harder to implement. Thus the permutation group that is returned is always simply-transitive.

The 'arithmetical' features (decomposition and ramification groups, Artin symbols etc) are only available for Galois fields.

---

artin_symbol(*P*)
Return the Artin symbol $\left(\frac{K/\mathbb{Q}}{\mathfrak{P}}\right)$, where K is the number field of self, and $\mathfrak{P}$ is an unramified prime ideal. This is the unique element s of the decomposition group of $\mathfrak{P}$ such that $s(x) = x^p \bmod \mathfrak{P}$, where p is the residue characteristic of $\mathfrak{P}$.

EXAMPLES:

```
sage: K.<b> = NumberField(x^4 - 2*x^2 + 2, 'a').galois_closure()
sage: G = K.galois_group()
sage: [G.artin_symbol(P) for P in K.primes_above(7)]
```

```
[(1,5)(2,6)(3,7)(4,8), (1,5)(2,6)(3,7)(4,8), (1,4)(2,3)(5,8)(6,7), (1,4)(2,3)(5,8)(6,7)]
sage: G.artin_symbol(17)
Traceback (most recent call last):
...
ValueError: Fractional ideal (17) is not prime
sage: QuadraticField(-7,'c').galois_group().artin_symbol(13)
(1,2)
sage: G.artin_symbol(K.primes_above(2)[0])
Traceback (most recent call last):
...
ValueError: Fractional ideal (...) is ramified
```

complex_conjugation(*P=None*)

Return the unique element of self corresponding to complex conjugation, for a specified embedding P into the complex numbers. If P is not specified, use the "standard" embedding, whenever that is well-defined.

EXAMPLE:

```
sage: L.<z> = CyclotomicField(7)
sage: G = L.galois_group()
sage: conj = G.complex_conjugation(); conj
(1,4)(2,5)(3,6)
sage: conj(z)
-z^5 - z^4 - z^3 - z^2 - z - 1
```

An example where the field is not CM, so complex conjugation really depends on the choice of embedding:

```
sage: L = NumberField(x^6 + 40*x^3 + 1372,'a')
sage: G = L.galois_group()
sage: [G.complex_conjugation(x) for x in L.places()]
[(1,3)(2,6)(4,5), (1,5)(2,4)(3,6), (1,2)(3,4)(5,6)]
```

decomposition_group(*P*)

Decomposition group of a prime ideal P, i.e. the subgroup of elements that map P to itself. This is the same as the Galois group of the extension of local fields obtained by completing at P.

This function will raise an error if P is not prime or the given number field is not Galois.

P can also be an infinite prime, i.e. an embedding into $\mathbf{R}$ or $\mathbf{C}$.

EXAMPLE:

```
sage: K.<a> = NumberField(x^4 - 2*x^2 + 2,'b').galois_closure()
sage: P = K.ideal([17, a^2])
sage: G = K.galois_group()
sage: G.decomposition_group(P)
Subgroup [(), (1,8)(2,7)(3,6)(4,5)] of Galois group of Number Field in a with defining polyr
sage: G.decomposition_group(P^2)
Traceback (most recent call last):
...
ValueError: Fractional ideal (...) is not prime
sage: G.decomposition_group(17)
Traceback (most recent call last):
...
ValueError: Fractional ideal (17) is not prime
```

An example with an infinite place:

```
sage: L.<b> = NumberField(x^3 - 2,'a').galois_closure(); G=L.galois_group()
sage: x = L.places()[0]
```

```
sage: G.decomposition_group(x).order()
2
```

inertia_group(*P*)

Return the inertia group of the prime P, i.e. the group of elements acting trivially modulo P. This is just the 0th ramification group of P.

EXAMPLE:

```
sage: K.<b> = NumberField(x^2 - 3,'a')
sage: G = K.galois_group()
sage: G.inertia_group(K.primes_above(2)[0])
Galois group of Number Field in b with defining polynomial x^2 - 3
sage: G.inertia_group(K.primes_above(5)[0])
Subgroup [()] of Galois group of Number Field in b with defining polynomial x^2 - 3
```

is_galois()

Return True if the underlying number field of self is actually Galois.

EXAMPLE:

```
sage: NumberField(x^3 - x + 1,'a').galois_group(names='b').is_galois()
False
sage: NumberField(x^2 - x + 1,'a').galois_group().is_galois()
True
```

list()

List of the elements of self.

EXAMPLE:

```
sage: NumberField(x^3 - 3*x + 1,'a').galois_group().list()
[(), (1,2,3), (1,3,2)]
```

ngens()

Number of generators of self.

EXAMPLE:

```
sage: QuadraticField(-23, 'a').galois_group().ngens()
1
```

number_field()

The ambient number field.

EXAMPLE:

```
sage: K = NumberField(x^3 - x + 1, 'a')
sage: K.galois_group(names='b').number_field() is K
True
```

ramification_breaks(*P*)

Return the set of ramification breaks of the prime ideal P, i.e. the set of indices i such that the ramification group $G_{i+1} \neq G_i$. This is only defined for Galois fields.

EXAMPLE:

```
sage: K.<b> = NumberField(x^8 - 20*x^6 + 104*x^4 - 40*x^2 + 1156)
sage: G = K.galois_group()
sage: P = K.primes_above(2)[0]
sage: G.ramification_breaks(P)
{1, 3, 5}
```

```
sage: min( [ G.ramification_group(P, i).order() / G.ramification_group(P, i+1).order() for i
2
```

ramification_group(*P*, *v*)

Return the vth ramification group of self for the prime P, i.e. the set of elements s of self such that s acts trivially modulo P^(v+1). This is only defined for Galois fields.

EXAMPLE:

```
sage: K.<b> = NumberField(x^3 - 3,'a').galois_closure()
sage: G=K.galois_group()
sage: P = K.primes_above(3)[0]
sage: G.ramification_group(P, 3)
Subgroup [(), (1,2,4)(3,5,6), (1,4,2)(3,6,5)] of Galois group of Number Field in b with defi
sage: G.ramification_group(P, 5)
Subgroup [()] of Galois group of Number Field in b with defining polynomial x^6 + 243
```

splitting_field()

The Galois closure of the ambient number field.

EXAMPLE:

```
sage: K = NumberField(x^3 - x + 1, 'a')
sage: K.galois_group(names='b').splitting_field()
Number Field in b with defining polynomial x^6 - 6*x^4 + 9*x^2 + 23
sage: L = QuadraticField(-23, 'c'); L.galois_group().splitting_field() is L
True
```

subgroup(*elts*)

Return the subgroup of self with the given elements. Mostly for internal use.

EXAMPLE:

```
sage: G = NumberField(x^3 - x - 1, 'a').galois_closure('b').galois_group()
sage: G.subgroup([ G(1), G([(1,2,3),(4,5,6)]), G([(1,3,2),(4,6,5)]) ])
Subgroup [(), (1,2,3)(4,5,6), (1,3,2)(4,6,5)] of Galois group of Number Field in b with defi
```

CHAPTER

FOURTEEN

UNIT AND S-UNIT GROUPS OF NUMBER FIELDS

EXAMPLES:

```
sage: x = polygen(QQ)
sage: K.<a> = NumberField(x^4-8*x^2+36)
sage: UK = UnitGroup(K); UK
Unit group with structure C4 x Z of Number Field in a with defining polynomial x^4 - 8*x^2 + 36
```

The first generator is a primitive root of unity in the field:

```
sage: UK.gens()
(u0, u1)
sage: UK.gens_values()  # random
[-1/12*a^3 + 1/6*a, 1/24*a^3 + 1/4*a^2 - 1/12*a - 1]
sage: UK.gen(0).value()
-1/12*a^3 + 1/6*a

sage: UK.gen(0)
u0
sage: UK.gen(0) + K.one()    # coerce abstract generator into number field
-1/12*a^3 + 1/6*a + 1

sage: [u.multiplicative_order() for u in UK.gens()]
[4, +Infinity]
sage: UK.rank()
1
sage: UK.ngens()
2
```

Units in the field can be converted into elements of the unit group represented as elements of an abstract multiplicative group:

```
sage: UK(1)
1
sage: UK(-1)
u0^2
sage: [UK(u) for u in (x^4-1).roots(K,multiplicities=False)]
[1, u0^2, u0^3, u0]

sage: UK.fundamental_units() # random
[1/24*a^3 + 1/4*a^2 - 1/12*a - 1]
sage: torsion_gen = UK.torsion_generator();  torsion_gen
u0
sage: torsion_gen.value()
-1/12*a^3 + 1/6*a
sage: UK.zeta_order()
```

```
4
sage: UK.roots_of_unity()
[-1/12*a^3 + 1/6*a, -1, 1/12*a^3 - 1/6*a, 1]
```

Exp and log functions provide maps between units as field elements and exponent vectors with respect to the generators:

```
sage: u = UK.exp([13,10]); u # random
-41/8*a^3 - 55/4*a^2 + 41/4*a + 55
sage: UK.log(u)
(1, 10)
sage: u = UK.fundamental_units()[0]
sage: [UK.log(u^k) == (0,k) for k in range(10)]
[True, True, True, True, True, True, True, True, True, True]
sage: all([UK.log(u^k) == (0,k) for k in range(10)])
True

sage: K.<a> = NumberField(x^5-2,'a')
sage: UK = UnitGroup(K)
sage: UK.rank()
2
sage: UK.fundamental_units()
[a^3 + a^2 - 1, a - 1]
```

S-unit groups may be constructed, where S is a set of primes:

```
sage: K.<a> = NumberField(x^6+2)
sage: S = K.ideal(3).prime_factors(); S
[Fractional ideal (3, a + 1), Fractional ideal (3, a - 1)]
sage: SUK = UnitGroup(K,S=tuple(S)); SUK
S-unit group with structure C2 x Z x Z x Z x Z of Number Field in a with defining polynomial x^6 + 2
sage: SUK.primes()
(Fractional ideal (3, a + 1), Fractional ideal (3, a - 1))
sage: SUK.rank()
4
sage: SUK.gens_values()
[-1, a^2 + 1, a^5 + a^4 - a^2 - a - 1, a + 1, -a + 1]
sage: u = 9*prod(SUK.gens_values()); u
-18*a^5 - 18*a^4 - 18*a^3 - 9*a^2 + 9*a + 27
sage: SUK.log(u)
(1, 3, 1, 7, 7)
sage: u == SUK.exp((1,3,1,7,7))
True
```

A relative number field example:

```
sage: L.<a, b> = NumberField([x^2 + x + 1, x^4 + 1])
sage: UL = L.unit_group(); UL
Unit group with structure C24 x Z x Z x Z of Number Field in a with defining polynomial x^2 + x + 1 o
sage: UL.gens_values() # random
[-b^3*a - b^3, -b^3*a + b, (-b^3 - b^2 - b)*a - b - 1, (-b^3 - 1)*a - b^2 + b - 1]
sage: UL.zeta_order()
24
sage: UL.roots_of_unity()
[b*a, -b^2*a - b^2, b^3, -a, b*a + b, -b^2, -b^3*a, -a - 1, b, b^2*a, -b^3*a - b^3, -1, -b*a, b^2*a -
```

A relative extension example, which worked thanks to the code review by F.W.Clarke:

```
sage: PQ.<X> = QQ[]
sage: F.<a, b> = NumberField([X^2 - 2, X^2 - 3])
sage: PF.<Y> = F[]
sage: K.<c> = F.extension(Y^2 - (1 + a)*(a + b)*a*b)
sage: K.unit_group()
Unit group with structure C2 x Z x Z x Z x Z x Z x Z x Z of Number Field in c with defining polynomia
```

TESTS:

```
sage: UK == loads(dumps(UK))
True
sage: UL == loads(dumps(UL))
True
```

AUTHOR:

- John Cremona

class sage.rings.number_field.unit_group.**UnitGroup**(*number_field*, *proof=True*, *S=None*)

Bases: sage.groups.abelian_gps.values.AbelianGroupWithValues_class

The unit group or an S-unit group of a number field.

TESTS:

```
sage: x = polygen(QQ)
sage: K.<a> = NumberField(x^4 + 23)
sage: UK = K.unit_group()
sage: u = UK.an_element();  u
u0*u1
sage: u.value()
-1/4*a^3 + 7/4*a^2 - 17/4*a + 19/4

sage: x = polygen(QQ)
sage: K.<a> = NumberField(x^4 + 23)
sage: K.unit_group().gens_values() # random
[-1, 1/4*a^3 - 7/4*a^2 + 17/4*a - 19/4]

sage: x = polygen(QQ)
sage: U = NumberField(x^2 + x + 23899, 'a').unit_group(); U
Unit group with structure C2 of Number Field in a with defining polynomial x^2 + x + 23899
sage: U.ngens()
1

sage: K.<z> = CyclotomicField(13)
sage: UK = K.unit_group()
sage: UK.ngens()
6
sage: UK.gen(5)
u5
sage: UK.gen(5).value()
z^7 + z
```

An S-unit group:

```
sage: SUK = UnitGroup(K,S=21); SUK
S-unit group with structure C26 x Z x Z x Z x Z x Z x Z x Z x Z x Z x Z x Z of Cyclotomic Field of c
sage: SUK.rank()
10
sage: SUK.zeta_order()
26
```

```
sage: SUK.log(21*z)
(12, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1)
```

exp(*exponents*)

Return unit with given exponents with respect to group generators.

INPUT:

- u – Any object from which an element of the unit group's number field K may be constructed; an error is raised if an element of K cannot be constructed from u, or if the element constructed is not a unit.

OUTPUT: a list of integers giving the exponents of u with respect to the unit group's basis.

EXAMPLES:

```
sage: x = polygen(QQ)
sage: K.<z> = CyclotomicField(13)
sage: UK = UnitGroup(K)
sage: [UK.log(u) for u in UK.gens()]
[(1, 0, 0, 0, 0, 0),
 (0, 1, 0, 0, 0, 0),
 (0, 0, 1, 0, 0, 0),
 (0, 0, 0, 1, 0, 0),
 (0, 0, 0, 0, 1, 0),
 (0, 0, 0, 0, 0, 1)]
sage: vec = [65,6,7,8,9,10]
sage: unit = UK.exp(vec)
sage: UK.log(unit)
(13, 6, 7, 8, 9, 10)
sage: UK.exp(UK.log(u)) == u.value()
True
```

An S-unit example:

```
sage: SUK = UnitGroup(K,S=2)
sage: v = (3,1,4,1,5,9,2)
sage: u = SUK.exp(v); u
-8732*z^11 + 15496*z^10 + 51840*z^9 + 68804*z^8 + 51840*z^7 + 15496*z^6 - 8732*z^5 + 34216*z
sage: SUK.log(u)
(3, 1, 4, 1, 5, 9, 2)
sage: SUK.log(u) == v
True
```

fundamental_units()

Return generators for the free part of the unit group, as a list.

EXAMPLES:

```
sage: x = polygen(QQ)
sage: K.<a> = NumberField(x^4 + 23)
sage: U = UnitGroup(K)
sage: U.fundamental_units()  # random
[1/4*a^3 - 7/4*a^2 + 17/4*a - 19/4]
```

log(*u*)

Return the exponents of the unit u with respect to group generators.

INPUT:

- u – Any object from which an element of the unit group's number field K may be constructed; an

error is raised if an element of K cannot be constructed from u, or if the element constructed is not a unit.

OUTPUT: a list of integers giving the exponents of u with respect to the unit group's basis.

EXAMPLES:

```
sage: x = polygen(QQ)
sage: K.<z> = CyclotomicField(13)
sage: UK = UnitGroup(K)
sage: [UK.log(u) for u in UK.gens()]
[(1, 0, 0, 0, 0, 0),
 (0, 1, 0, 0, 0, 0),
 (0, 0, 1, 0, 0, 0),
 (0, 0, 0, 1, 0, 0),
 (0, 0, 0, 0, 1, 0),
 (0, 0, 0, 0, 0, 1)]
sage: vec = [65,6,7,8,9,10]
sage: unit = UK.exp(vec); unit  # random
-253576*z^11 + 7003*z^10 - 395532*z^9 - 35275*z^8 - 500326*z^7 - 35275*z^6 - 395532*z^5 + 70
sage: UK.log(unit)
(13, 6, 7, 8, 9, 10)
```

An S-unit example:

```
sage: SUK = UnitGroup(K,S=2)
sage: v = (3,1,4,1,5,9,2)
sage: u = SUK.exp(v); u
-8732*z^11 + 15496*z^10 + 51840*z^9 + 68804*z^8 + 51840*z^7 + 15496*z^6 - 8732*z^5 + 34216*z
sage: SUK.log(u)
(3, 1, 4, 1, 5, 9, 2)
sage: SUK.log(u) == v
True
```

number_field()

Return the number field associated with this unit group.

EXAMPLES:

```
sage: U = UnitGroup(QuadraticField(-23, 'w')); U
Unit group with structure C2 of Number Field in w with defining polynomial x^2 + 23
sage: U.number_field()
Number Field in w with defining polynomial x^2 + 23
```

primes()

Return the (possibly empty) list of primes associated with this S-unit group.

EXAMPLES:

```
sage: K.<a> = QuadraticField(-23)
sage: S = tuple(K.ideal(3).prime_factors()); S
(Fractional ideal (3, 1/2*a - 1/2), Fractional ideal (3, 1/2*a + 1/2))
sage: U = UnitGroup(K,S=tuple(S)); U
S-unit group with structure C2 x Z x Z of Number Field in a with defining polynomial x^2 + 2
sage: U.primes() == S
True
```

rank()

Return the rank of the unit group.

EXAMPLES:

```
sage: K.<z> = CyclotomicField(13)
sage: UnitGroup(K).rank()
5
sage: SUK = UnitGroup(K,S=2); SUK.rank()
6
```

roots_of_unity()

Return all the roots of unity in this unit group, primitive or not.

EXAMPLES:

```
sage: x = polygen(QQ)
sage: K.<b> = NumberField(x^2+1)
sage: U = UnitGroup(K)
sage: zs = U.roots_of_unity(); zs
[b, -1, -b, 1]
sage: [ z**U.zeta_order() for z in zs ]
[1, 1, 1, 1]
```

torsion_generator()

Return a generator for the torsion part of the unit group.

EXAMPLES:

```
sage: x = polygen(QQ)
sage: K.<a> = NumberField(x^4 - x^2 + 4)
sage: U = UnitGroup(K)
sage: U.torsion_generator()
u0
sage: U.torsion_generator().value() # random
-1/4*a^3 - 1/4*a + 1/2
```

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

Return one, or a list of all, primitive n-th root of unity in this unit group.

EXAMPLES:

```
sage: x = polygen(QQ)
sage: K.<z> = NumberField(x^2 + 3)
sage: U = UnitGroup(K)
sage: U.zeta(1)
1
sage: U.zeta(2)
-1
sage: U.zeta(2, all=True)
[-1]
sage: U.zeta(3)
-1/2*z - 1/2
sage: U.zeta(3, all=True)
[-1/2*z - 1/2, 1/2*z - 1/2]
sage: U.zeta(4)
Traceback (most recent call last):
...
ValueError: n (=4) does not divide order of generator

sage: r.<x> = QQ[]
sage: K.<b> = NumberField(x^2+1)
sage: U = UnitGroup(K)
sage: U.zeta(4)
b
```

```
sage: U.zeta(4,all=True)
[b, -b]
sage: U.zeta(3)
Traceback (most recent call last):
...
ValueError: n (=3) does not divide order of generator
sage: U.zeta(3,all=True)
[]
```

zeta_order()

Returns the order of the torsion part of the unit group.

EXAMPLES:

```
sage: x = polygen(QQ)
sage: K.<a> = NumberField(x^4 - x^2 + 4)
sage: U = UnitGroup(K)
sage: U.zeta_order()
6
```

CHAPTER

FIFTEEN

SMALL PRIMES OF DEGREE ONE

Iterator for finding several primes of absolute degree one of a number field of *small* prime norm.

Algorithm:

Let P denote the product of some set of prime numbers. (In practice, we use the product of the first 10000 primes, because Pari computes this many by default.)

Let K be a number field and let $f(x)$ be a polynomial defining K over the rational field. Let α be a root of f in K.

We know that $[O_K : \mathbf{Z}[\alpha]]^2 = |\Delta(f(x))/\Delta(O_K)|$, where Δ denotes the discriminant (see, for example, Proposition 4.4.4, p165 of [C]). Therefore, after discarding primes dividing $\Delta(f(x))$ (this includes all ramified primes), any integer n such that $\gcd(f(n), P) > 0$ yields a prime $p|P$ such that $f(x)$ has a root modulo p. By the condition on discriminants, this root is a single root. As is well known (see, for example Theorem 4.8.13, p199 of [C]), the ideal generated by $(p, \alpha - n)$ is prime and of degree one.

Warning: It is possible that there are no primes of K of absolute degree one of small prime norm, and it is possible that this algorithm will not find any primes of small norm.

To do:

There are situations when this will fail. There are questions of finding primes of relative degree one. There are questions of finding primes of exact degree larger than one. In short, if you can contribute, please do!

EXAMPLES:

```
sage: x = ZZ['x'].gen()
sage: F.<a> = NumberField(x^2 - 2)
sage: Ps = F.primes_of_degree_one_list(3)
sage: Ps # random
[Fractional ideal (2*a + 1), Fractional ideal (-3*a + 1), Fractional ideal (-a + 5)]
sage: [ P.norm() for P in Ps ] # random
[7, 17, 23]
sage: all(ZZ(P.norm()).is_prime() for P in Ps)
True
sage: all(P.residue_class_degree() == 1 for P in Ps)
True
```

The next two examples are for relative number fields.:

```
sage: L.<b> = F.extension(x^3 - a)
sage: Ps = L.primes_of_degree_one_list(3)
sage: Ps # random
[Fractional ideal (17, b - 5), Fractional ideal (23, b - 4), Fractional ideal (31, b - 2)]
sage: [ P.absolute_norm() for P in Ps ] # random
[17, 23, 31]
sage: all(ZZ(P.absolute_norm()).is_prime() for P in Ps)
True
sage: all(P.residue_class_degree() == 1 for P in Ps)
True
sage: M.<c> = NumberField(x^2 - x*b^2 + b)
sage: Ps = M.primes_of_degree_one_list(3)
sage: Ps # random
[Fractional ideal (17, c - 2), Fractional ideal (c - 1), Fractional ideal (41, c + 15)]
sage: [ P.absolute_norm() for P in Ps ] # random
[17, 31, 41]
sage: all(ZZ(P.absolute_norm()).is_prime() for P in Ps)
True
sage: all(P.residue_class_degree() == 1 for P in Ps)
True
```

AUTHORS:

- Nick Alexander (2008)
- David Loeffler (2009): fixed a bug with relative fields

class sage.rings.number_field.small_primes_of_degree_one.**Small_primes_of_degree_one_iter**(*field*, *num_*
*max_*

Iterator that finds primes of a number field of absolute degree one and bounded small prime norm.

INPUT:

- `field` – a `NumberField`.
- `num_integer_primes` (default: 10000) – an integer. We try to find primes of absolute norm no greater than the `num_integer_primes`-th prime number. For example, if `num_integer_primes` is 2, the largest norm found will be 3, since the second prime is 3.
- `max_iterations` (default: 100) – an integer. We test `max_iterations` integers to find small primes before raising `StopIteration`.

AUTHOR:

- Nick Alexander

next()

Return a prime of absolute degree one of small prime norm.

Raises `StopIteration` if such a prime cannot be easily found.

EXAMPLES:

```
sage: x = QQ['x'].gen()
sage: K.<a> = NumberField(x^2 - 3)
sage: it = K.primes_of_degree_one_iter()
sage: [ next(it) for i in range(3) ] # random
[Fractional ideal (2*a + 1), Fractional ideal (-a + 4), Fractional ideal (3*a + 2)]
```

We test that #6396 is fixed. Note that the doctest is flagged as random since the string representation of ideals is somewhat unpredictable:

```
sage: N.<a,b> = NumberField([x^2 + 1, x^2 - 5])
sage: ids = N.primes_of_degree_one_list(10); ids  # random
[Fractional ideal ((-1/2*b + 1/2)*a + 2),
 Fractional ideal (-b*a + 1/2*b + 1/2),
 Fractional ideal ((1/2*b + 3/2)*a - b),
 Fractional ideal ((-1/2*b - 3/2)*a + b - 1),
 Fractional ideal (-b*a - b + 1),
 Fractional ideal (3*a + 1/2*b - 1/2),
 Fractional ideal ((-3/2*b + 1/2)*a + 1/2*b - 1/2),
 Fractional ideal ((-1/2*b - 5/2)*a - b + 1),
 Fractional ideal (2*a - 3/2*b - 1/2),
 Fractional ideal (3*a + 1/2*b + 5/2)]
 sage: [x.absolute_norm() for x in ids]
 [29, 41, 61, 89, 101, 109, 149, 181, 229, 241]
 sage: ids[9] == N.ideal(3*a + 1/2*b + 5/2)
 True
```

CHAPTER

SIXTEEN

SPLITTING FIELDS OF POLYNOMIALS OVER NUMBER FIELDS

AUTHORS:

- Jeroen Demeyer (2014-01-02): initial version for trac ticket #2217
- Jeroen Demeyer (2014-01-03): add `abort_degree` argument, trac ticket #15626

class sage.rings.number_field.splitting_field.**SplittingData**(*_pol*, *_dm*)

A class to store data for internal use in `splitting_field()`. It contains two attributes `pol` (polynomial), `dm` (degree multiple), where `pol` is a PARI polynomial and `dm` a Sage `Integer`.

`dm` is a multiple of the degree of the splitting field of `pol` over some field E. In `splitting_field()`, E is the field containing the current field K and all roots of other polynomials inside the list L with `dm` less than this `dm`.

key()

Return a sorting key. Compare first by degree bound, then by polynomial degree, then by discriminant.

EXAMPLES:

```
sage: from sage.rings.number_field.splitting_field import SplittingData
sage: L = []
sage: L.append(SplittingData(pari("x^2 + 1"), 1))
sage: L.append(SplittingData(pari("x^3 + 1"), 1))
sage: L.append(SplittingData(pari("x^2 + 7"), 2))
sage: L.append(SplittingData(pari("x^3 + 1"), 2))
sage: L.append(SplittingData(pari("x^3 + x^2 + x + 1"), 2))
sage: L.sort(key=lambda x: x.key()); L
[SplittingData(x^2 + 1, 1), SplittingData(x^3 + 1, 1), SplittingData(x^2 + 7, 2), SplittingD
sage: [x.key() for x in L]
[(1, 2, 16), (1, 3, 729), (2, 2, 784), (2, 3, 256), (2, 3, 729)]
```

poldegree()

Return the degree of `self.pol`

EXAMPLES:

```
sage: from sage.rings.number_field.splitting_field import SplittingData
sage: SplittingData(pari("x^123 + x + 1"), 2).poldegree()
123
```

exception sage.rings.number_field.splitting_field.**SplittingFieldAbort**(*div*, *mult*)

Bases: `exceptions.Exception`

Special exception class to indicate an early abort of `splitting_field()`.

EXAMPLES:

```
sage: from sage.rings.number_field.splitting_field import SplittingFieldAbort
sage: raise SplittingFieldAbort(20, 60)
Traceback (most recent call last):
...
SplittingFieldAbort: degree of splitting field is a multiple of 20
sage: raise SplittingFieldAbort(12, 12)
Traceback (most recent call last):
...
SplittingFieldAbort: degree of splitting field equals 12
```

sage.rings.number_field.splitting_field.**splitting_field**(*poly*, *name*, *map=False*, *degree_multiple=None*, *abort_degree=None*, *simplify=True*, *simplify_all=False*)

Compute the splitting field of a given polynomial, defined over a number field.

INPUT:

- `poly` – a monic polynomial over a number field
- `name` – a variable name for the number field
- `map` – (default: `False`) also return an embedding of `poly` into the resulting field. Note that computing this embedding might be expensive.
- `degree_multiple` – a multiple of the absolute degree of the splitting field. If `degree_multiple` equals the actual degree, this can enormously speed up the computation.
- `abort_degree` – abort by raising a `SplittingFieldAbort` if it can be determined that the absolute degree of the splitting field is strictly larger than `abort_degree`.
- `simplify` – (default: `True`) during the algorithm, try to find a simpler defining polynomial for the intermediate number fields using PARI's `polred()`. This usually speeds up the computation but can also considerably slow it down. Try and see what works best in the given situation.
- `simplify_all` – (default: `False`) If `True`, simplify intermediate fields and also the resulting number field.

OUTPUT:

If `map` is `False`, the splitting field as an absolute number field. If `map` is `True`, a tuple `(K, phi)` where `phi` is an embedding of the base field in `K`.

EXAMPLES:

```
sage: R.<x> = PolynomialRing(QQ)
sage: K.<a> = (x^3 + 2).splitting_field(); K
Number Field in a with defining polynomial x^6 + 3*x^5 + 6*x^4 + 11*x^3 + 12*x^2 - 3*x + 1
sage: K.<a> = (x^3 - 3*x + 1).splitting_field(); K
Number Field in a with defining polynomial x^3 - 3*x + 1
```

The `simplify` and `simplify_all` flags usually yield fields defined by polynomials with smaller coefficients. By default, `simplify` is True and `simplify_all` is False.

```
sage: (x^4 - x + 1).splitting_field('a', simplify=False)
Number Field in a with defining polynomial x^24 - 2780*x^22 + 2*x^21 + 3527512*x^20 - 2876*x^19
sage: (x^4 - x + 1).splitting_field('a', simplify=True)
Number Field in a with defining polynomial x^24 + 8*x^23 - 32*x^22 - 310*x^21 + 540*x^20 + 4688*
sage: (x^4 - x + 1).splitting_field('a', simplify_all=True)
Number Field in a with defining polynomial x^24 - 3*x^23 + 2*x^22 - x^20 + 4*x^19 + 32*x^18 - 35
```

Reducible polynomials also work:

```
sage: pol = (x^4 - 1)*(x^2 + 1/2)*(x^2 + 1/3)
sage: pol.splitting_field('a', simplify_all=True)
Number Field in a with defining polynomial x^8 - x^4 + 1
```

Relative situation:

```
sage: R.<x> = PolynomialRing(QQ)
sage: K.<a> = NumberField(x^3 + 2)
sage: S.<t> = PolynomialRing(K)
sage: L.<b> = (t^2 - a).splitting_field()
sage: L
Number Field in b with defining polynomial t^6 + 2
```

With `map=True`, we also get the embedding of the base field into the splitting field:

```
sage: L.<b>, phi = (t^2 - a).splitting_field(map=True)
sage: phi
Ring morphism:
  From: Number Field in a with defining polynomial x^3 + 2
  To:   Number Field in b with defining polynomial t^6 + 2
  Defn: a |--> b^2
sage: (x^4 - x + 1).splitting_field('a', simplify_all=True, map=True)[1]
Ring morphism:
  From: Rational Field
  To:   Number Field in a with defining polynomial x^24 - 3*x^23 + 2*x^22 - x^20 + 4*x^19 + 32*x
  Defn: 1 |--> 1
```

We can enable verbose messages:

```
sage: set_verbose(2)
sage: K.<a> = (x^3 - x + 1).splitting_field()
verbose 1 (...: splitting_field.py, splitting_field) Starting field: y
verbose 1 (...: splitting_field.py, splitting_field) SplittingData to factor: [(3, 0)]
verbose 2 (...: splitting_field.py, splitting_field) Done factoring (time = ...)
verbose 1 (...: splitting_field.py, splitting_field) SplittingData to handle: [(2, 2), (3, 3)]
verbose 1 (...: splitting_field.py, splitting_field) Bounds for absolute degree: [6, 6]
verbose 2 (...: splitting_field.py, splitting_field) Handling polynomial x^2 + 23
verbose 1 (...: splitting_field.py, splitting_field) New field before simplifying: x^2 + 23 (tim
verbose 1 (...: splitting_field.py, splitting_field) New field: y^2 - y + 6 (time = ...)
verbose 2 (...: splitting_field.py, splitting_field) Converted polynomials to new field (time =
verbose 1 (...: splitting_field.py, splitting_field) SplittingData to factor: []
verbose 2 (...: splitting_field.py, splitting_field) Done factoring (time = ...)
verbose 1 (...: splitting_field.py, splitting_field) SplittingData to handle: [(3, 3)]
verbose 1 (...: splitting_field.py, splitting_field) Bounds for absolute degree: [6, 6]
verbose 2 (...: splitting_field.py, splitting_field) Handling polynomial x^3 - x + 1
verbose 1 (...: splitting_field.py, splitting_field) New field: y^6 + 3*y^5 + 19*y^4 + 35*y^3 +
sage: set_verbose(0)
```

Try all Galois groups in degree 4. We use a quadratic base field such that `polgalois()` cannot be used:

```
sage: R.<x> = PolynomialRing(QuadraticField(-11))
sage: C2C2pol = x^4 - 10*x^2 + 1
sage: C2C2pol.splitting_field('x')
Number Field in x with defining polynomial x^8 + 24*x^6 + 608*x^4 + 9792*x^2 + 53824
sage: C4pol = x^4 + x^3 + x^2 + x + 1
sage: C4pol.splitting_field('x')
Number Field in x with defining polynomial x^8 - x^7 - 2*x^6 + 5*x^5 + x^4 + 15*x^3 - 18*x^2 - 2
sage: D8pol = x^4 - 2
sage: D8pol.splitting_field('x')
```

```
Number Field in x with defining polynomial x^16 + 8*x^15 + 68*x^14 + 336*x^13 + 1514*x^12 + 5080
sage: A4pol = x^4 - 4*x^3 + 14*x^2 - 28*x + 21
sage: A4pol.splitting_field('x')
Number Field in x with defining polynomial x^24 - 20*x^23 + 290*x^22 - 3048*x^21 + 26147*x^20 -
sage: S4pol = x^4 + x + 1
sage: S4pol.splitting_field('x')
Number Field in x with defining polynomial x^48 ...
```

Some bigger examples:

```
sage: R.<x> = PolynomialRing(QQ)
sage: pol15 = chebyshev_T(31, x) - 1    # 2^30*(x-1)*minpoly(cos(2*pi/31))^2
sage: pol15.splitting_field('a')
Number Field in a with defining polynomial x^15 - x^14 - 14*x^13 + 13*x^12 + 78*x^11 - 66*x^10 -
sage: pol48 = x^6 - 4*x^4 + 12*x^2 - 12
sage: pol48.splitting_field('a')
Number Field in a with defining polynomial x^48 ...
```

If you somehow know the degree of the field in advance, you should add a `degree_multiple` argument. This can speed up the computation, in particular for polynomials of degree >= 12 or for relative extensions:

```
sage: pol15.splitting_field('a', degree_multiple=15)
Number Field in a with defining polynomial x^15 + x^14 - 14*x^13 - 13*x^12 + 78*x^11 + 66*x^10 -
```

A value for `degree_multiple` which isn't actually a multiple of the absolute degree of the splitting field can either result in a wrong answer or the following exception:

```
sage: pol48.splitting_field('a', degree_multiple=20)
Traceback (most recent call last):
...
ValueError: inconsistent degree_multiple in splitting_field()
```

Compute the Galois closure as the splitting field of the defining polynomial:

```
sage: R.<x> = PolynomialRing(QQ)
sage: pol48 = x^6 - 4*x^4 + 12*x^2 - 12
sage: K.<a> = NumberField(pol48)
sage: L.<b> = pol48.change_ring(K).splitting_field()
sage: L
Number Field in b with defining polynomial x^48 ...
```

Try all Galois groups over $\mathbf{Q}$ in degree 5 except for S_5 (the latter is infeasible with the current implementation):

```
sage: C5pol = x^5 + x^4 - 4*x^3 - 3*x^2 + 3*x + 1
sage: C5pol.splitting_field('x')
Number Field in x with defining polynomial x^5 + x^4 - 4*x^3 - 3*x^2 + 3*x + 1
sage: D10pol = x^5 - x^4 - 5*x^3 + 4*x^2 + 3*x - 1
sage: D10pol.splitting_field('x')
Number Field in x with defining polynomial x^10 - 28*x^8 + 216*x^6 - 681*x^4 + 902*x^2 - 401
sage: AGL_1_5pol = x^5 - 2
sage: AGL_1_5pol.splitting_field('x')
Number Field in x with defining polynomial x^20 + 10*x^19 + 55*x^18 + 210*x^17 + 595*x^16 + 1300
sage: A5pol = x^5 - x^4 + 2*x^2 - 2*x + 2
sage: A5pol.splitting_field('x')
Number Field in x with defining polynomial x^60 ...
```

We can use the `abort_degree` option if we don't want to compute fields of too large degree (this can be used to check whether the splitting field has small degree):

```
sage: (x^5+x+3).splitting_field('b', abort_degree=119)
Traceback (most recent call last):
...
SplittingFieldAbort: degree of splitting field equals 120
sage: (x^10+x+3).splitting_field('b', abort_degree=60)  # long time (10s on sage.math, 2014)
Traceback (most recent call last):
...
SplittingFieldAbort: degree of splitting field is a multiple of 180
```

Use the `degree_divisor` attribute to recover the divisor of the degree of the splitting field or `degree_multiple` to recover a multiple:

```
sage: from sage.rings.number_field.splitting_field import SplittingFieldAbort
sage: try:  # long time (4s on sage.math, 2014)
....:     (x^8+x+1).splitting_field('b', abort_degree=60, simplify=False)
....: except SplittingFieldAbort as e:
....:     print e.degree_divisor
....:     print e.degree_multiple
120
1440
```

TESTS:

```
sage: from sage.rings.number_field.splitting_field import splitting_field
sage: splitting_field(polygen(QQ), name='x', map=True, simplify_all=True)
(Number Field in x with defining polynomial x, Ring morphism:
  From: Rational Field
  To:   Number Field in x with defining polynomial x
  Defn: 1 |--> 1)
```

CHAPTER

SEVENTEEN

ELEMENTS OF BOUNDED HEIGHT IN NUMBER FIELDS

Sage functions to list all elements of a given number field with height less than a specified bound.

AUTHORS:

- John Doyle (2013): initial version
- David Krumm (2013): initial version

REFERENCES:

sage.rings.number_field.bdd_height.**bdd_height**(*K*, *height_bound*, *precision=53*, *LLL=False*)

Computes all elements in the number number field K which have relative multiplicative height at most `height_bound`.

The algorithm requires arithmetic with floating point numbers; `precision` gives the user the option to set the precision for such computations.

It might be helpful to work with an LLL-reduced system of fundamental units, so the user has the option to perform an LLL reduction for the fundamental units by setting `LLL` to True.

Certain computations may be faster assuming GRH, which may be done globally by using the number_field(True/False) switch.

The function will only be called for number fields K with positive unit rank. An error will occur if K is QQ or an imaginary quadratic field.

ALGORITHM:

This is an implementation of the main algorithm (Algorithm 3) in [Doyle-Krumm].

INPUT:

•`height_bound` - real number

•`precision` - (default: 53) positive integer

•`LLL` - (default: False) boolean value

OUTPUT:

•an iterator of number field elements

Warning: In the current implementation, the output of the algorithm cannot be guaranteed to be correct due to the necessity of floating point computations. In some cases, the default 53-bit precision is considerably lower than would be required for the algorithm to generate correct output.

Todo

Should implement a version of the algorithm that guarantees correct output. See Algorithm 4 in [Doyle-Krumm] for details of an implementation that takes precision issues into account.

EXAMPLES:

There are no elements of negative height:

```
sage: from sage.rings.number_field.bdd_height import bdd_height
sage: K.<g> = NumberField(x^5 - x + 7)
sage: list(bdd_height(K,-3))
[]
```

The only nonzero elements of height 1 are the roots of unity:

```
sage: from sage.rings.number_field.bdd_height import bdd_height
sage: K.<g> = QuadraticField(3)
sage: list(bdd_height(K,1))
[0, -1, 1]

sage: from sage.rings.number_field.bdd_height import bdd_height
sage: K.<g> = QuadraticField(36865)
sage: len(list(bdd_height(K,101))) # long time (4 s)
131

sage: from sage.rings.number_field.bdd_height import bdd_height
sage: K.<g> = NumberField(x^3 - 197*x + 39)
sage: len(list(bdd_height(K, 200))) # long time (5 s)
451

sage: from sage.rings.number_field.bdd_height import bdd_height
sage: K.<g> = NumberField(x^6 + 2)
sage: len(list(bdd_height(K,60,precision=100))) # long time (5 s)
1899

sage: from sage.rings.number_field.bdd_height import bdd_height
sage: K.<g> = NumberField(x^4 - x^3 - 3*x^2 + x + 1)
sage: len(list(bdd_height(K,10,LLL=true)))
99
```

sage.rings.number_field.bdd_height.**bdd_height_iq**(*K*, *height_bound*)

Compute all elements in the imaginary quadratic field K which have relative multiplicative height at most `height_bound`.

The function will only be called with K an imaginary quadratic field.

If called with K not an imaginary quadratic, the function will likely yield incorrect output.

ALGORITHM:

This is an implementation of Algorithm 5 in [Doyle-Krumm].

INPUT:

- K - an imaginary quadratic number field
- `height_bound` - a real number

OUTPUT:

- an iterator of number field elements

EXAMPLES:

```
sage: from sage.rings.number_field.bdd_height import bdd_height_iq
sage: K.<a> = NumberField(x^2 + 191)
sage: for t in bdd_height_iq(K,8):
....:     print exp(2*t.global_height())
....:
1.00000000000000
1.00000000000000
1.00000000000000
4.00000000000000
4.00000000000000
4.00000000000000
4.00000000000000
8.00000000000000
8.00000000000000
8.00000000000000
8.00000000000000
8.00000000000000
8.00000000000000
8.00000000000000
8.00000000000000
```

There are 175 elements of height at most 10 in $QQ(\sqrt{(}-3))$:

```
sage: from sage.rings.number_field.bdd_height import bdd_height_iq
sage: K.<a> = NumberField(x^2 + 3)
sage: len(list(bdd_height_iq(K,10)))
175
```

The only elements of multiplicative height 1 in a number field are 0 and the roots of unity:

```
sage: from sage.rings.number_field.bdd_height import bdd_height_iq
sage: K.<a> = NumberField(x^2 + x + 1)
sage: list(bdd_height_iq(K,1))
[0, a + 1, a, -1, -a - 1, -a, 1]
```

A number field has no elements of multiplicative height less than 1:

```
sage: from sage.rings.number_field.bdd_height import bdd_height_iq
sage: K.<a> = NumberField(x^2 + 5)
sage: list(bdd_height_iq(K,0.9))
[]
```

sage.rings.number_field.bdd_height.**bdd_norm_pr_gens_iq**(*K*, *norm_list*)

Compute generators for all principal ideals in an imaginary quadratic field K whose norms are in `norm_list`.

The only keys for the output dictionary are integers n appearing in `norm_list`.

The function will only be called with K an imaginary quadratic field.

The function will return a dictionary for other number fields, but it may be incorrect.

INPUT:

- K - an imaginary quadratic number field
- `norm_list` - a list of positive integers

OUTPUT:

- a dictionary of number field elements, keyed by norm

EXAMPLES:

In $QQ(i)$, there is one principal ideal of norm 4, two principal ideals of norm 5, but no principal ideals of norm 7:

```
sage: from sage.rings.number_field.bdd_height import bdd_norm_pr_gens_iq
sage: K.<g> = NumberField(x^2 + 1)
sage: L = range(10)
sage: bdd_pr_ideals = bdd_norm_pr_gens_iq(K, L)
sage: bdd_pr_ideals[4]
[2]
sage: bdd_pr_ideals[5]
[g + 2, g - 2]
sage: bdd_pr_ideals[7]
[]
```

There are no ideals in the ring of integers with negative norm:

```
sage: from sage.rings.number_field.bdd_height import bdd_norm_pr_gens_iq
sage: K.<g> = NumberField(x^2 + 10)
sage: L = range(-5,-1)
sage: bdd_pr_ideals = bdd_norm_pr_gens_iq(K,L)
sage: bdd_pr_ideals
{-5: [], -4: [], -3: [], -2: []}
```

Calling a key that is not in the input `norm_list` raises a KeyError:

```
sage: from sage.rings.number_field.bdd_height import bdd_norm_pr_gens_iq
sage: K.<g> = NumberField(x^2 + 20)
sage: L = range(100)
sage: bdd_pr_ideals = bdd_norm_pr_gens_iq(K, L)
sage: bdd_pr_ideals[100]
Traceback (most recent call last):
...
KeyError: 100
```

sage.rings.number_field.bdd_height.**bdd_norm_pr_ideal_gens**(*K*, *norm_list*)

Compute generators for all principal ideals in a number field K whose norms are in `norm_list`.

INPUT:

- K - a number field
- `norm_list` - a list of positive integers

OUTPUT:

- a dictionary of number field elements, keyed by norm

EXAMPLES:

There is only one principal ideal of norm 1, and it is generated by the element 1:

```
sage: from sage.rings.number_field.bdd_height import bdd_norm_pr_ideal_gens
sage: K.<g> = QuadraticField(101)
sage: bdd_norm_pr_ideal_gens(K, [1])
{1: [1]}
```

```
sage: from sage.rings.number_field.bdd_height import bdd_norm_pr_ideal_gens
sage: K.<g> = QuadraticField(123)
sage: bdd_norm_pr_ideal_gens(K, range(5))
{0: [0], 1: [1], 2: [-g - 11], 3: [], 4: [2]}
```

```
sage: from sage.rings.number_field.bdd_height import bdd_norm_pr_ideal_gens
sage: K.<g> = NumberField(x^5 - x + 19)
sage: b = bdd_norm_pr_ideal_gens(K, range(30))
sage: key = ZZ(28)
sage: b[key]
[157*g^4 - 139*g^3 - 369*g^2 + 848*g + 158, g^4 + g^3 - g - 7]
```

sage.rings.number_field.bdd_height.**integer_points_in_polytope**(*matrix*, *interval_radius*)

Return the set of integer points in the polytope obtained by acting on a cube by a linear transformation.

Given an r-by-r matrix `matrix` and a real number `interval_radius`, this function finds all integer lattice points in the polytope obtained by transforming the cube [-interval_radius,interval_radius]^r via the linear map induced by `matrix`.

INPUT:

- `matrix` - a square matrix of real numbers
- `interval_radius` - a real number

OUTPUT:

- a list of tuples of integers

EXAMPLES:

Stretch the interval [-1,1] by a factor of 2 and find the integers in the resulting interval:

```
sage: from sage.rings.number_field.bdd_height import integer_points_in_polytope
sage: m = matrix([2])
sage: r = 1
sage: integer_points_in_polytope(m,r)
[(-2), (-1), (0), (1), (2)]
```

Integer points inside a parallelogram:

```
sage: from sage.rings.number_field.bdd_height import integer_points_in_polytope
sage: m = matrix([[1, 2],[3, 4]])
sage: r = RealField()(1.3)
sage: integer_points_in_polytope(m,r)
[(-3, -7), (-2, -5), (-2, -4), (-1, -3), (-1, -2), (-1, -1), (0, -1), (0, 0), (0, 1), (1, 1), (1
```

Integer points inside a parallelepiped:

```
sage: from sage.rings.number_field.bdd_height import integer_points_in_polytope
sage: m = matrix([[1.2,3.7,0.2],[-5.3,-.43,3],[1.2,4.7,-2.1]])
sage: r = 2.2
sage: L = integer_points_in_polytope(m,r)
sage: len(L)
4143
```

If `interval_radius` is 0, the output should include only the zero tuple:

```
sage: from sage.rings.number_field.bdd_height import integer_points_in_polytope
sage: m = matrix([[1,2,3,7],[4,5,6,2],[7,8,9,3],[0,3,4,5]])
sage: integer_points_in_polytope(m,0)
[(0, 0, 0, 0)]
```

CHAPTER

EIGHTEEN

HELPER CLASSES FOR STRUCTURAL EMBEDDINGS AND ISOMORPHISMS OF NUMBER FIELDS

AUTHORS:

- Julian Rueth (2014-04-03): initial version

Consider the following fields L and M:

```
sage: L.<a> = QuadraticField(2)
sage: M.<a> = L.absolute_field()
```

Both produce the same extension of $\mathbf{Q}$. However, they should not be identical because M carries additional information:

```
sage: L.structure()
(Ring Coercion endomorphism of Number Field in a with defining polynomial x^2 - 2,
 Ring Coercion endomorphism of Number Field in a with defining polynomial x^2 - 2)
sage: M.structure()
(Isomorphism given by variable name change map:
  From: Number Field in a with defining polynomial x^2 - 2
  To:   Number Field in a with defining polynomial x^2 - 2,
 Isomorphism given by variable name change map:
  From: Number Field in a with defining polynomial x^2 - 2
  To:   Number Field in a with defining polynomial x^2 - 2)
```

This used to cause trouble with caching and made (absolute) number fields not unique when they should have been. The underlying technical problem is that the morphisms returned by `structure()` can only be defined once the fields in question have been created. Therefore, these morphisms cannot be part of a key which uniquely identifies a number field.

The classes defined in this file encapsulate information about these structure morphisms which can be passed to the factory creating number fields. This makes it possible to distinguish number fields which only differ in terms of these structure morphisms:

```
sage: L is M
False
sage: N.<a> = L.absolute_field()
sage: M is N
True
```

class sage.rings.number_field.structure.**AbsoluteFromRelative**(*other*)

Bases: sage.rings.number_field.structure.NumberFieldStructureFromUniqueField

Structure for an absolute number field created from a relative number field.

INPUT:

- other – the number field from which this field has been created.

TESTS:

```
sage: from sage.rings.number_field.structure import AbsoluteFromRelative
sage: K.<a> = QuadraticField(2)
sage: R.<x> = K[]
sage: L.<b> = K.extension(x^2 - 3)
sage: AbsoluteFromRelative(L)
<sage.rings.number_field.structure.AbsoluteFromRelative object at 0x...>
```

create_structure(*field*)

Return a pair of isomorphisms which go from field to other and vice versa.

TESTS:

```
sage: K.<a> = QuadraticField(2)
sage: R.<x> = K[]
sage: L.<b> = K.extension(x^2 - 3)
sage: M.<c> = L.absolute_field()
sage: M.structure() # indirect doctest
(Isomorphism map:
  From: Number Field in c with defining polynomial x^4 - 10*x^2 + 1
  To:   Number Field in b with defining polynomial x^2 - 3 over its base field, Isomorphism
  From: Number Field in b with defining polynomial x^2 - 3 over its base field
  To:   Number Field in c with defining polynomial x^4 - 10*x^2 + 1)
```

class sage.rings.number_field.structure.**NameChange**(*other*)

Bases: sage.rings.number_field.structure.NumberFieldStructureFromUniqueField

Structure for a number field created by a change in variable name.

INPUT:

- other – the number field from which this field has been created.

TESTS:

```
sage: from sage.rings.number_field.structure import NameChange
sage: K.<i> = QuadraticField(-1)
sage: NameChange(K)
<sage.rings.number_field.structure.NameChange object at 0x...>
```

create_structure(*field*)

Return a pair of isomorphisms which send the generator of field to the generator of other and vice versa.

TESTS:

```
sage: CyclotomicField(5).absolute_field('a').structure() # indirect doctest
(Isomorphism given by variable name change map:
  From: Number Field in a with defining polynomial x^4 + x^3 + x^2 + x + 1
  To:   Cyclotomic Field of order 5 and degree 4,
 Isomorphism given by variable name change map:
  From: Cyclotomic Field of order 5 and degree 4
  To:   Number Field in a with defining polynomial x^4 + x^3 + x^2 + x + 1)
```

class sage.rings.number_field.structure.**NumberFieldStructure**

Bases: object

Abstract base class encapsulating information about a number fields relation to other number fields.

TESTS:

```
sage: from sage.rings.number_field.structure import NumberFieldStructure
sage: NumberFieldStructure()
<sage.rings.number_field.structure.NumberFieldStructure object at 0x...>
```

create_structure(*field*)

Return a tuple encoding structural information about `field`.

OUTPUT:

Typically, the output is a pair of morphisms. The first one from `field` to a field from which `field` has been constructed and the second one its inverse. In this case, these morphisms are used as conversion maps between the two fields.

TESTS:

```
sage: from sage.rings.number_field.structure import NumberFieldStructure
sage: NumberFieldStructure().create_structure(QQ)
Traceback (most recent call last):
...
NotImplementedError
```

The morphisms created by this method are used as conversion maps:

```
sage: K.<i> = QuadraticField(-1)
sage: L.<j> = K.change_names()
sage: isinstance(L._structure, NumberFieldStructure)
True
sage: from_L, to_L = L.structure()
sage: L._convert_map_from_(K) is to_L
True
sage: L(i)
j
sage: K(j)
i
```

class sage.rings.number_field.structure.**NumberFieldStructureFromUniqueField**(*other*)

Bases: sage.structure.unique_representation.UniqueRepresentation, sage.rings.number_field.structure.NumberFieldStructure

Abstract base class encapsulating information about a number fields relation to another number field.

INPUT:

- `other` – a number field

TESTS:

sage: from sage.rings.number_field.structure import NumberFieldStructureFromUniqueField sage: NumberFieldStructureFromUniqueField(QQ) <sage.rings.number_field.structure.NumberFieldStructureFromUniqueField object at 0x...>

Instances are cached through `sage.structure.unique_representation.UniqueRepresentation`:

```
sage: NumberFieldStructureFromUniqueField(QQ) is NumberFieldStructureFromUniqueField(QQ)
True
```

However, the caching (and the == operator of the objects) is on identity and not on equality:

```
sage: R.<x> = QQ[]
sage: K.<i> = NumberField(x^2+1)
sage: L = K.change_names('j').change_names('i')
sage: K == L
```

```
True
sage: K is L # K and L differ in "structure", one is the "name-change" of the other
False

sage: NumberFieldStructureFromUniqueField(L) is NumberFieldStructureFromUniqueField(L)
True
sage: NumberFieldStructureFromUniqueField(K) is NumberFieldStructureFromUniqueField(L)
False
```

This important because otherwise caching of number fields would be broken:

```
sage: R.<x> = QQ[]
sage: from sage.rings.number_field.structure import NameChange
sage: KK.<j> = NumberField(x^2+1, structure=NameChange(K))
sage: LL.<j> = NumberField(x^2+1, structure=NameChange(L))
sage: KK is LL
False
```

class sage.rings.number_field.structure.**RelativeFromAbsolute**(*other*, *gen*)

Bases: sage.rings.number_field.structure.NumberFieldStructureFromUniqueField

Structure for a relative number field created from an absolute number field.

INPUT:

- `other` – the (absolute) number field from which this field has been created.
- `gen` – the generator of the intermediate field

TESTS:

```
sage: from sage.rings.number_field.structure import RelativeFromAbsolute
sage: RelativeFromAbsolute(QQ, 1/2)
<sage.rings.number_field.structure.RelativeFromAbsolute object at 0x...>
```

create_structure(*field*)

Return a pair of isomorphisms which go from `field` to `other` and vice versa.

INPUT:

- `field` – a relative number field

TESTS:

```
sage: K.<a> = QuadraticField(2)
sage: M.<b,a_> = K.relativize(-a)
sage: M.structure() # indirect doctest
(Relative number field morphism:
 From: Number Field in b with defining polynomial x + a_ over its base field
 To:   Number Field in a with defining polynomial x^2 - 2
 Defn: -a_ |--> a
       a_ |--> -a, Ring morphism:
 From: Number Field in a with defining polynomial x^2 - 2
 To:   Number Field in b with defining polynomial x + a_ over its base field
 Defn: a |--> -a_)
```

class sage.rings.number_field.structure.**RelativeFromRelative**(*other*)

Bases: sage.rings.number_field.structure.NumberFieldStructureFromUniqueField

Structure for a relative number field created from another relative number field.

INPUT:

- `other` – the relative number field used in the construction, see `create_structure()`; there this field will be called `field_`.

TESTS:

```
sage: from sage.rings.number_field.structure import RelativeFromRelative
sage: K.<i> = QuadraticField(-1)
sage: R.<x> = K[]
sage: L.<a> = K.extension(x^2 - 2)
sage: RelativeFromRelative(L)
<sage.rings.number_field.structure.RelativeFromRelative object at 0x...>
```

create_structure(*field*)

Return a pair of isomorphisms which go from `field` to the relative number field (called `other` below) from which `field` has been created and vice versa.

The isomorphism is created via the relative number field `field_` which is identical to `field` but is equipped with an isomorphism to an absolute field which was used in the construction of `field`.

INPUT:

- `field` – a relative number field

TESTS:

```
sage: K.<i> = QuadraticField(-1)
sage: R.<x> = K[]
sage: L.<a> = K.extension(x^2 - 2)
sage: M.<b,a> = L.relativize(a)
sage: M.structure() # indirect doctest
(Relative number field morphism:
  From: Number Field in b with defining polynomial x^2 - 2*a*x + 3 over its base field
  To:   Number Field in a with defining polynomial x^2 - 2 over its base field
  Defn: b |--> a - i
        a |--> a, Relative number field morphism:
  From: Number Field in a with defining polynomial x^2 - 2 over its base field
  To:   Number Field in b with defining polynomial x^2 - 2*a*x + 3 over its base field
  Defn: a |--> a
        i |--> -b + a)
```

CHAPTER

NINETEEN

ENUMERATION OF PRIMITIVE TOTALLY REAL FIELDS

This module contains functions for enumerating all primitive totally real number fields of given degree and small discriminant. Here a number field is called *primitive* if it contains no proper subfields except $\mathbf{Q}$.

See also `sage.rings.number_field.totallyreal_rel`, which handles the non-primitive case using relative extensions.

19.1 Algorithm

We use Hunter's algorithm ([Cohen2000], Section 9.3) with modifications due to Takeuchi [Takeuchi1999] and the author [Voight2008].

We enumerate polynomials $f(x) = x^n + a_{n-1}x^{n-1} + \cdots + a_0$. Hunter's theorem gives bounds on a_{n-1} and a_{n-2}; then given a_{n-1} and a_{n-2}, one can recursively compute bounds on $a_{n-3}, \ldots, a_0$, using the fact that the polynomial is totally real by looking at the zeros of successive derivatives and applying Rolle's theorem. See [Takeuchi1999] for more details.

19.2 Examples

In this first simple example, we compute the totally real quadratic fields of discriminant ≤ 50.

```
sage: enumerate_totallyreal_fields_prim(2,50)
[[5, x^2 - x - 1],
 [8, x^2 - 2],
 [12, x^2 - 3],
 [13, x^2 - x - 3],
 [17, x^2 - x - 4],
 [21, x^2 - x - 5],
 [24, x^2 - 6],
 [28, x^2 - 7],
 [29, x^2 - x - 7],
 [33, x^2 - x - 8],
 [37, x^2 - x - 9],
 [40, x^2 - 10],
 [41, x^2 - x - 10],
 [44, x^2 - 11]]
sage: [ d for d in range(5,50) if (is_squarefree(d) and d%4 == 1) or (d%4 == 0 and is_squarefree(d/4)
[5, 8, 12, 13, 17, 20, 21, 24, 28, 29, 33, 37, 40, 41, 44]
```

Next, we compute all totally real quintic fields of discriminant $\leq 10^5$:

```
sage: ls = enumerate_totallyreal_fields_prim(5,10^5) ; ls
[[14641, x^5 - x^4 - 4*x^3 + 3*x^2 + 3*x - 1],
 [24217, x^5 - 5*x^3 - x^2 + 3*x + 1],
 [36497, x^5 - 2*x^4 - 3*x^3 + 5*x^2 + x - 1],
 [38569, x^5 - 5*x^3 + 4*x - 1],
 [65657, x^5 - x^4 - 5*x^3 + 2*x^2 + 5*x + 1],
 [70601, x^5 - x^4 - 5*x^3 + 2*x^2 + 3*x - 1],
 [81509, x^5 - x^4 - 5*x^3 + 3*x^2 + 5*x - 2],
 [81589, x^5 - 6*x^3 + 8*x - 1],
 [89417, x^5 - 6*x^3 - x^2 + 8*x + 3]]
 sage: len(ls)
 9
```

We see that there are 9 such fields (up to isomorphism!).

19.3 References

19.4 Authors

- John Voight (2007-09-01): Initial version.
- John Voight (2007-09-19): Various optimization tweaks.
- John Voight (2007-10-09): Added DSage module.
- John Voight (2007-10-17): Added pari functions to avoid recomputations.
- John Voight (2007-10-27): Separated DSage component.
- Craig Citro and John Voight (2007-11-04): Additional doctests and type checking.
- Craig Citro and John Voight (2008-02-10): Final modifications for submission.

---

sage.rings.number_field.totallyreal.**enumerate_totallyreal_fields_prim**(*n*, *B*, *a*=[], *verbose=0*, *return_seqs=False*, *phc=False*, *keep_fields=False*, *t_2=False*, *just_print=False*, *return_pari_objects=True*)

This function enumerates primitive totally real fields of degree $n > 1$ with discriminant $d \leq B$; optionally one can specify the first few coefficients, where the sequence a corresponds to
`a[d]*x^n + ... + a[0]*x^(n-d)`

where `length(a) = d+1`, so in particular always `a[d] = 1`.

Note: This is guaranteed to give all primitive such fields, and seems in practice to give many imprimitive ones.

INPUT:

- `n` – (integer) the degree
- `B` – (integer) the discriminant bound
- `a` – (list, default: []) the coefficient list to begin with
- `verbose` – (integer or string, default: 0) if `verbose == 1` (or 2), then print to the screen (really) verbosely; if verbose is a string, then print verbosely to the file specified by verbose.
- `return_seqs` – (boolean, default False) If `True`, then return the polynomials as sequences (for easier exporting to a file).
- `phc` – boolean or integer (default: False)
- `keep_fields` – (boolean or integer, default: False) If `keep_fields` is True, then keep fields up to `B*log(B)`; if `keep_fields` is an integer, then keep fields up to that integer.
- `t_2` – (boolean or integer, default: False) If `t_2 = T`, then keep only polynomials with t_2 norm >= T.
- `just_print` – (boolean, default: False): if `just_print` is not False, instead of creating a sorted list of totally real number fields, we simply write each totally real field we find to the file whose filename is given by `just_print`. In this case, we don't return anything.
- `return_pari_objects` – (boolean, default: True) if both `return_seqs` and `return_pari_objects` are `False` then it returns the elements as Sage objects; otherwise it returns pari objects.

OUTPUT:

the list of fields with entries `[d,f]`, where `d` is the discriminant and `f` is a defining polynomial, sorted by discriminant.

AUTHORS:

- John Voight (2007-09-03)
- Craig Citro (2008-09-19): moved to Cython for speed improvement

TESTS:

```
sage: len(enumerate_totallyreal_fields_prim(2,10**4))
3043
sage: len(enumerate_totallyreal_fields_prim(3,3**8))
237
sage: len(enumerate_totallyreal_fields_prim(5,5**7))
6
sage: len(enumerate_totallyreal_fields_prim(2,2**15)) # long time
9957
sage: len(enumerate_totallyreal_fields_prim(3,3**10)) # long time
2720
sage: len(enumerate_totallyreal_fields_prim(5,5**8)) # long time
103
```

Each of the outputs must be elements of Sage if `return_pari_objects` is set to `False`:

```
sage: enumerate_totallyreal_fields_prim(2, 10)
[[5, x^2 - x - 1], [8, x^2 - 2]]
sage: enumerate_totallyreal_fields_prim(2, 10)[0][1].parent()
Interface to the PARI C library
sage: enumerate_totallyreal_fields_prim(2, 10, return_pari_objects=False)[0][0].parent()
Integer Ring
sage: enumerate_totallyreal_fields_prim(2, 10, return_pari_objects=False)[0][1].parent()
Univariate Polynomial Ring in x over Rational Field
```

```
sage: enumerate_totallyreal_fields_prim(2, 10, return_seqs=True)[1][0][1][0].parent()
Rational Field
```

sage.rings.number_field.totallyreal.**odlyzko_bound_totallyreal**(*n*)

This function returns the unconditional Odlyzko bound for the root discriminant of a totally real number field of degree n.

Note: The bounds for n > 50 are not necessarily optimal.

INPUT:

- n (integer) the degree

OUTPUT:

a lower bound on the root discriminant (as a real number)

EXAMPLES:

```
sage: [sage.rings.number_field.totallyreal.odlyzko_bound_totallyreal(n) for n in range(1,5)]
[1.0, 2.223, 3.61, 5.067]
```

AUTHORS:

- John Voight (2007-09-03)

NOTES: The values are calculated by Martinet [Martinet1980].

sage.rings.number_field.totallyreal.**timestr**(*m*)

Converts seconds to a human-readable time string.

INPUT:

- m – integer, number of seconds

OUTPUT:

The time in days, hours, etc.

EXAMPLES:

```
sage: sage.rings.number_field.totallyreal.timestr(3765)
'1h 2m 45.0s'
```

sage.rings.number_field.totallyreal.**weed_fields**(*S*, *lenS=0*)

Function used internally by the `enumerate_totallyreal_fields_prim()` routine. (Weeds the fields listed by [discriminant, polynomial] for isomorphism classes.) Returns the size of the resulting list.

EXAMPLES:

```
sage: ls = [[5,pari('x^2-3*x+1')],[5,pari('x^2-5')]]
sage: sage.rings.number_field.totallyreal.weed_fields(ls)
1
sage: ls
[[5, x^2 - 3*x + 1]]
```

CHAPTER

TWENTY

ENUMERATION OF TOTALLY REAL FIELDS: RELATIVE EXTENSIONS

This module contains functions to enumerate primitive extensions L/K, where K is a given totally real number field, with given degree and small root discriminant. This is a relative analogue of the problem described in `sage.rings.number_field.totallyreal`, and we use a similar approach based on a relative version of Hunter's theorem.

In this first simple example, we compute the totally real quadratic fields of $F = \mathbf{Q}(\sqrt{2})$ of discriminant ≤ 2000.

```
sage: ZZx = ZZ['x']
sage: F.<t> = NumberField(x^2-2)
sage: enumerate_totallyreal_fields_rel(F, 2, 2000)
[[1600, x^4 - 6*x^2 + 4, xF^2 + xF - 1]]
```

There is indeed only one such extension, given by $F(\sqrt{5})$.

Next, we list all totally real quadratic extensions of $\mathbf{Q}(\sqrt{5})$ with root discriminant ≤ 10.

```
sage: F.<t> = NumberField(x^2-5)
sage: ls = enumerate_totallyreal_fields_rel(F, 2, 10^4)
sage: ls # random (the second factor is platform-dependent)
[[725, x^4 - x^3 - 3*x^2 + x + 1, xF^2 + (-1/2*t - 7/2)*xF + 1],
 [1125, x^4 - x^3 - 4*x^2 + 4*x + 1, xF^2 + (-1/2*t - 7/2)*xF + 1/2*t + 3/2],
 [1600, x^4 - 6*x^2 + 4, xF^2 - 2],
 [2000, x^4 - 5*x^2 + 5, xF^2 - 1/2*t - 5/2],
 [2225, x^4 - x^3 - 5*x^2 + 2*x + 4, xF^2 + (-1/2*t + 1/2)*xF - 3/2*t - 7/2],
 [2525, x^4 - 2*x^3 - 4*x^2 + 5*x + 5, xF^2 + (-1/2*t - 1/2)*xF - 1/2*t - 5/2],
 [3600, x^4 - 2*x^3 - 7*x^2 + 8*x + 1, xF^2 - 3],
 [4225, x^4 - 9*x^2 + 4, xF^2 + (-1/2*t - 1/2)*xF - 3/2*t - 9/2],
 [4400, x^4 - 7*x^2 + 11, xF^2 - 1/2*t - 7/2],
 [4525, x^4 - x^3 - 7*x^2 + 3*x + 9, xF^2 + (-1/2*t - 1/2)*xF - 3],
 [5125, x^4 - 2*x^3 - 6*x^2 + 7*x + 11, xF^2 + (-1/2*t - 1/2)*xF - t - 4],
 [5225, x^4 - x^3 - 8*x^2 + x + 11, xF^2 + (-1/2*t - 1/2)*xF - 1/2*t - 7/2],
 [5725, x^4 - x^3 - 8*x^2 + 6*x + 11, xF^2 + (-1/2*t + 1/2)*xF - 1/2*t - 7/2],
 [6125, x^4 - x^3 - 9*x^2 + 9*x + 11, xF^2 + (-1/2*t + 1/2)*xF - t - 4],
 [7225, x^4 - 11*x^2 + 9, xF^2 + (-1)*xF - 4],
 [7600, x^4 - 9*x^2 + 19, xF^2 - 1/2*t - 9/2],
 [7625, x^4 - x^3 - 9*x^2 + 4*x + 16, xF^2 + (-1/2*t - 1/2)*xF - 4],
 [8000, x^4 - 10*x^2 + 20, xF^2 - t - 5],
 [8525, x^4 - 2*x^3 - 8*x^2 + 9*x + 19, xF^2 + (-1)*xF - 1/2*t - 9/2],
 [8725, x^4 - x^3 - 10*x^2 + 2*x + 19, xF^2 + (-1/2*t - 1/2)*xF - 1/2*t - 9/2],
 [9225, x^4 - x^3 - 10*x^2 + 7*x + 19, xF^2 + (-1/2*t + 1/2)*xF - 1/2*t - 9/2]]
sage: [ f[0] for f in ls ]
[725, 1125, 1600, 2000, 2225, 2525, 3600, 4225, 4400, 4525, 5125, 5225, 5725, 6125, 7225, 7600, 7625,

sage: [NumberField(ZZx(x[1]), 't').is_galois() for x in ls]
[False, True, True, True, False, False, True, True, False, False, False, False, False, True, True, Fa
```

Eight out of 21 such fields are Galois (with Galois group C_4 or $C_2 \times C_2$); the others have have Galois closure of degree 8 (with Galois group D_8).

Finally, we compute the cubic extensions of $\mathbf{Q}(\zeta_7)^+$ with discriminant $\leq 17 \times 10^9$.

```
sage: F.<t> = NumberField(ZZx([1,-4,3,1]))
sage: F.disc()
49
sage: enumerate_totallyreal_fields_rel(F, 3, 17*10^9)  # not tested, too long time (258s on sage.mat
[[16240385609L, x^9 - x^8 - 9*x^7 + 4*x^6 + 26*x^5 - 2*x^4 - 25*x^3 - x^2 + 7*x + 1, xF^3 + (-t^2 - 4
[[16240385609, x^9 - x^8 - 9*x^7 + 4*x^6 + 26*x^5 - 2*x^4 - 25*x^3 - x^2 + 7*x + 1, xF^3 + (-t^2 - 4
```

AUTHORS:

- John Voight (2007-11-03): Initial version.

sage.rings.number_field.totallyreal_rel.**enumerate_totallyreal_fields_all**(*n*, *B*, *verbose=0*, *return_seqs=False*, *return_pari_objects=True*)

Enumerates *all* totally real fields of degree `n` with discriminant at most `B`, primitive or otherwise.

INPUT:

- `n` – integer, the degree
- `B` – integer, the discriminant bound
- `verbose` – boolean or nonnegative integer or string (default: 0) give a verbose description of the computations being performed. If `verbose` is set to `2` or more then it outputs some extra information. If `verbose` is a string then it outputs to a file specified by `verbose`
- `return_seqs` – (boolean, default False) If `True`, then return the polynomials as sequences (for easier exporting to a file). This also returns a list of four numbers, as explained in the OUTPUT section below.
- `return_pari_objects` – (boolean, default: True) if both `return_seqs` and `return_pari_objects` are `False` then it returns the elements as Sage objects; otherwise it returns pari objects.

EXAMPLES:

```
sage: enumerate_totallyreal_fields_all(4, 2000)
[[725, x^4 - x^3 - 3*x^2 + x + 1],
[1125, x^4 - x^3 - 4*x^2 + 4*x + 1],
[1600, x^4 - 6*x^2 + 4],
[1957, x^4 - 4*x^2 - x + 1],
[2000, x^4 - 5*x^2 + 5]]
sage: enumerate_totallyreal_fields_all(1, 10)
[[1, x - 1]]
```

TESTS:

Each of the outputs must be elements of Sage if `return_pari_objects` is set to `False`:

```
sage: enumerate_totallyreal_fields_all(2, 10)
[[5, x^2 - x - 1], [8, x^2 - 2]]
sage: enumerate_totallyreal_fields_all(2, 10)[0][1].parent()
Interface to the PARI C library
```

```
sage: enumerate_totallyreal_fields_all(2, 10, return_pari_objects=False)[0][1].parent()
Univariate Polynomial Ring in x over Rational Field
```

In practice most of these will be found by `enumerate_totallyreal_fields_prim()`, which is guaranteed to return all primitive fields but often returns many non-primitive ones as well. For instance, only one of the five fields in the example above is primitive, but `enumerate_totallyreal_fields_prim()` finds four out of the five (the exception being $x^4 - 6x^2 + 4$).

The following was fixed in trac ticket #13101:

```
sage: enumerate_totallyreal_fields_all(8, 10^6)  # long time (about 2 s)
[]
```

sage.rings.number_field.totallyreal_rel.**enumerate_totallyreal_fields_rel**(*F*, *m*, *B*, *a=[]*, *verbose=0*, *return_seqs=False*, *return_pari_objects=True*)

This function enumerates (primitive) totally real field extensions of degree $m > 1$ of the totally real field F with discriminant $d \leq B$; optionally one can specify the first few coefficients, where the sequence `a` corresponds to a polynomial by

```
a[d]*x^n + ... + a[0]*x^(n-d)
```

if `length(a) = d+1`, so in particular always `a[d] = 1`.

Note: This is guaranteed to give all primitive such fields, and seems in practice to give many imprimitive ones.

INPUT:

- `F` – number field, the base field
- `m` – integer, the degree
- `B` – integer, the discriminant bound
- `a` – list (default: []), the coefficient list to begin with
- `verbose` – boolean or nonnegative integer or string (default: 0) give a verbose description of the computations being performed. If `verbose` is set to `2` or more then it outputs some extra information. If `verbose` is a string then it outputs to a file specified by `verbose`
- `return_seqs` – (boolean, default False) If `True`, then return the polynomials as sequences (for easier exporting to a file). This also returns a list of four numbers, as explained in the OUTPUT section below.
- `return_pari_objects` – (boolean, default: True) if both `return_seqs` and `return_pari_objects` are `False` then it returns the elements as Sage objects; otherwise it returns pari objects.

OUTPUT:

- the list of fields with entries `[d,fabs,f]`, where `d` is the discriminant, `fabs` is an absolute defining polynomial, and `f` is a defining polynomial relative to `F`, sorted by discriminant.

- if `return_seqs` is `True`, then the first field of the list is a list containing the count of four items as explained below
  - the first entry gives the number of polynomials tested
  - the second entry gives the number of polynomials with its discriminant having a large enough square divisor
  - the third entry is the number of irreducible polynomials
  - the fourth entry is the number of irreducible polynomials with discriminant at most `B`

EXAMPLES:

```
sage: ZZx = ZZ['x']
sage: F.<t> = NumberField(x^2-2)
sage: enumerate_totallyreal_fields_rel(F, 1, 2000)
[[1, [-2, 0, 1], xF - 1]]
sage: enumerate_totallyreal_fields_rel(F, 2, 2000)
[[1600, x^4 - 6*x^2 + 4, xF^2 + xF - 1]]
sage: enumerate_totallyreal_fields_rel(F, 2, 2000, return_seqs=True)
[[9, 6, 5, 0], [[1600, [4, 0, -6, 0, 1], [-1, 1, 1]]]]
```

TESTS:

Each of the outputs must be elements of Sage if `return_pari_objects` is set to `False`:

```
sage: enumerate_totallyreal_fields_rel(F, 2, 2000)[0][1].parent()
Interface to the PARI C library
sage: enumerate_totallyreal_fields_rel(F, 2, 2000, return_pari_objects=False)[0][0].parent()
Integer Ring
sage: enumerate_totallyreal_fields_rel(F, 2, 2000, return_pari_objects=False)[0][1].parent()
Univariate Polynomial Ring in x over Rational Field
sage: enumerate_totallyreal_fields_rel(F, 2, 2000, return_pari_objects=False)[0][2].parent()
Univariate Polynomial Ring in xF over Number Field in t with defining polynomial x^2 - 2
sage: enumerate_totallyreal_fields_rel(F, 2, 2000, return_seqs=True)[1][0][1][0].parent()
Rational Field
```

AUTHORS:

- John Voight (2007-11-01)

sage.rings.number_field.totallyreal_rel.**integral_elements_in_box**(*K*, *C*)

Return all integral elements of the totally real field K whose embeddings lie *numerically* within the bounds specified by the list C. The output is architecture dependent, and one may want to expand the bounds that define C by some epsilon.

INPUT:

- K – a totally real number field
- C – a list [[lower, upper], ...] of lower and upper bounds, for each embedding

EXAMPLES:

```
sage: x = polygen(QQ)
sage: K.<alpha> = NumberField(x^2-2)
sage: eps = 10e-6
sage: C = [[0-eps,5+eps],[0-eps,10+eps]]
sage: ls = sage.rings.number_field.totallyreal_rel.integral_elements_in_box(K, C)
sage: sorted([ a.trace() for a in ls ])
[0, 2, 4, 4, 4, 6, 6, 6, 6, 8, 8, 8, 10, 10, 10, 10, 12, 12, 14]
sage: len(ls)
19
```

```
sage: v = sage.rings.number_field.totallyreal_rel.integral_elements_in_box(K, C)
sage: sorted(v)
 [0, -alpha + 2, 1, -alpha + 3, 2, 3, alpha + 2, 4, alpha + 3, 5, alpha + 4, 2*alpha + 3, alpha
```

A cubic field:

```
sage: x = polygen(QQ)
sage: K.<a> = NumberField(x^3 - 16*x +16)
sage: eps = 10e-6
sage: C = [[0-eps,5+eps]]*3
sage: v = sage.rings.number_field.totallyreal_rel.integral_elements_in_box(K, C)
```

Note that the output is platform dependent (sometimes a 5 is listed below, and sometimes it isn't):

```
sage: sorted(v)
[-1/2*a + 2, 1/4*a^2 + 1/2*a, 0, 1, 2, 3, 4,...-1/4*a^2 - 1/2*a + 5, 1/2*a + 3, -1/4*a^2 + 5]
```

class sage.rings.number_field.totallyreal_rel.**tr_data_rel**(*F*, *m*, *B*, *a=None*)

This class encodes the data used in the enumeration of totally real fields for relative extensions.

We do not give a complete description here. For more information, see the attached functions; all of these are used internally by the functions in totallyreal_rel.py, so see that file for examples and further documentation.

incr(*f_out*, *verbose=False*, *haltk=0*)

This function 'increments' the totally real data to the next value which satisfies the bounds essentially given by Rolle's theorem, and returns the next polynomial in the sequence f_out.

The default or usual case just increments the constant coefficient; then inductively, if this is outside of the bounds we increment the next higher coefficient, and so on.

If there are no more coefficients to be had, returns the zero polynomial.

INPUT:

- `f_out` – an integer sequence, to be written with the coefficients of the next polynomial
- `verbose` – boolean or nonnegative integer (default: False) print verbosely computational details. It prints extra information if `verbose` is set to `2` or more
- `haltk` – integer, the level at which to halt the inductive coefficient bounds

OUTPUT:

the successor polynomial as a coefficient list.

CHAPTER

TWENTYONE

ENUMERATION OF TOTALLY REAL FIELDS

AUTHORS:

- Craig Citro and John Voight (2007-11-04): Type checking and other polishing.
- John Voight (2007-10-09): Improvements: Smyth bound, Lagrange multipliers for b.
- John Voight (2007-09-19): Various optimization tweaks.
- John Voight (2007-09-01): Initial version.

sage.rings.number_field.totallyreal_data.**easy_is_irreducible_py**(*f*)

Used solely for testing easy_is_irreducible.

EXAMPLES:

```
sage: sage.rings.number_field.totallyreal_data.easy_is_irreducible_py(pari('x^2+1'))
1
sage: sage.rings.number_field.totallyreal_data.easy_is_irreducible_py(pari('x^2-1'))
0
```

sage.rings.number_field.totallyreal_data.**hermite_constant**(*n*)

This function returns the nth Hermite constant

The nth Hermite constant (typically denoted γ_n), is defined to be

$$\max_L \min_{0 \neq x \in L} ||x||^2$$

where L runs over all lattices of dimension n and determinant 1.

For $n \leq 8$ it returns the exact value of γ_n, and for $n > 9$ it returns an upper bound on γ_n.

INPUT:

•n – integer

OUTPUT:

•(an upper bound for) the Hermite constant gamma_n

EXAMPLES:

```
sage: hermite_constant(1) # trivial one-dimensional lattice
1.0
sage: hermite_constant(2) # Eisenstein lattice
1.1547005383792515
sage: 2/sqrt(3.)
1.15470053837925
sage: hermite_constant(8) # E_8
2.0
```

Note: The upper bounds used can be found in [CS] and [CE].

REFERENCES:

AUTHORS:

- John Voight (2007-09-03)

sage.rings.number_field.totallyreal_data.**int_has_small_square_divisor**(*d*)

Returns the largest a such that a^2 divides d and a has prime divisors < 200.

EXAMPLES:

```
sage: from sage.rings.number_field.totallyreal_data import int_has_small_square_divisor
sage: int_has_small_square_divisor(500)
100
sage: is_prime(691)
True
sage: int_has_small_square_divisor(691)
1
sage: int_has_small_square_divisor(691^2)
1
```

sage.rings.number_field.totallyreal_data.**lagrange_degree_3**(*n*, *an1*, *an2*, *an3*)

Private function. Solves the equations which arise in the Lagrange multiplier for degree 3: for each 1 <= r <= n-2, we solve

r*x^i + (n-1-r)*y^i + z^i = s_i (i = 1,2,3)

where the s_i are the power sums determined by the coefficients a. We output the largest value of z which occurs. We use a precomputed elimination ideal.

EXAMPLES:

```
sage: ls = sage.rings.number_field.totallyreal_data.lagrange_degree_3(3,0,1,2)
sage: [RealField(10)(x) for x in ls]
[-1.0, -1.0]
sage: sage.rings.number_field.totallyreal_data.lagrange_degree_3(3,6,1,2) # random
[-5.8878, -5.8878]
```

TESTS:

Check that trac ticket #13101 is solved:

```
sage: sage.rings.number_field.totallyreal_data.lagrange_degree_3(4,12,19,42)
[0.0, -1.0]
```

class sage.rings.number_field.totallyreal_data.**tr_data**

Bases: object

This class encodes the data used in the enumeration of totally real fields.

We do not give a complete description here. For more information, see the attached functions; all of these are used internally by the functions in totallyreal.py, so see that file for examples and further documentation.

increment(*verbose=False*, *haltk=0*, *phc=False*)

This function 'increments' the totally real data to the next value which satisfies the bounds essentially given by Rolle's theorem, and returns the next polynomial as a sequence of integers.

The default or usual case just increments the constant coefficient; then inductively, if this is outside of the bounds we increment the next higher coefficient, and so on.

If there are no more coefficients to be had, returns the zero polynomial.

INPUT:

- verbose – boolean to print verbosely computational details
- haltk – integer, the level at which to halt the inductive coefficient bounds
- phc – boolean, if PHCPACK is available, use it when k == n-5 to compute an improved Lagrange multiplier bound

OUTPUT:

The next polynomial, as a sequence of integers

EXAMPLES:

```
sage: T = sage.rings.number_field.totallyreal_data.tr_data(2,100)
sage: T.increment()
[-24, -1, 1]
sage: for i in range(19): _ = T.increment()
sage: T.increment()
[-3, -1, 1]
sage: T.increment()
[-25, 0, 1]
```

printa()

Print relevant data for self.

EXAMPLES:

```
sage: T = sage.rings.number_field.totallyreal_data.tr_data(3,2^10)
sage: T.printa()
k = 1
a = [0, 0, -1, 1]
amax = [0, 0, 0, 1]
beta =  [...]
gnk =  [...]
```

CHAPTER

TWENTYTWO

ENUMERATION OF TOTALLY REAL FIELDS: PHC INTERFACE

AUTHORS:

– John Voight (2007-10-10):

- Zeroth attempt.

sage.rings.number_field.totallyreal_phc.**coefficients_to_power_sums**(*n*, *m*, *a*)

Takes the list a, representing a list of initial coefficients of a (monic) polynomial of degree n, and returns the power sums of the roots of f up to (m-1)th powers.

INPUT:

•n – integer, the degree

•a – list of integers, the coefficients

OUTPUT:

list of integers.

NOTES:

Uses Newton's relations, which are classical.

AUTHORS:

•John Voight (2007-09-19)

EXAMPLES:

```
sage: sage.rings.number_field.totallyreal_phc.coefficients_to_power_sums(3,2,[1,5,7])
[3, -7, 39]
sage: sage.rings.number_field.totallyreal_phc.coefficients_to_power_sums(5,4,[1,5,7,9,8])
[5, -8, 46, -317, 2158]
```

CHAPTER

TWENTYTHREE

FIELD OF ALGEBRAIC NUMBERS

AUTHOR:

- Carl Witty (2007-01-27): initial version
- Carl Witty (2007-10-29): massive rewrite to support complex as well as real numbers

This is an implementation of the algebraic numbers (the complex numbers which are the zero of a polynomial in $\mathbf{Z}[x]$; in other words, the algebraic closure of $\mathbf{Q}$, with an embedding into $\mathbf{C}$). All computations are exact. We also include an implementation of the algebraic reals (the intersection of the algebraic numbers with $\mathbf{R}$). The field of algebraic numbers $\overline{\mathbf{Q}}$ is available with abbreviation `QQbar`; the field of algebraic reals has abbreviation `AA`.

As with many other implementations of the algebraic numbers, we try hard to avoid computing a number field and working in the number field; instead, we use floating-point interval arithmetic whenever possible (basically whenever we need to prove non-equalities), and resort to symbolic computation only as needed (basically to prove equalities).

Algebraic numbers exist in one of the following forms:

- a rational number
- the product of a rational number and an n'th root of unity
- the sum, difference, product, or quotient of algebraic numbers
- the negation, inverse, absolute value, norm, real part, imaginary part, or complex conjugate of an algebraic number
- a particular root of a polynomial, given as a polynomial with algebraic coefficients together with an isolating interval (given as a `RealIntervalFieldElement`) which encloses exactly one root, and the multiplicity of the root
- a polynomial in one generator, where the generator is an algebraic number given as the root of an irreducible polynomial with integral coefficients and the polynomial is given as a `NumberFieldElement`.

The multiplicative subgroup of the algebraic numbers generated by the rational numbers and the roots of unity is handled particularly efficiently, as long as these roots of unity come from the `QQbar.zeta()` method. Cyclotomic fields in general are fairly efficient, again as long as they are derived from `QQbar.zeta()`.

An algebraic number can be coerced into `ComplexIntervalField` (or `RealIntervalField`, for algebraic reals); every algebraic number has a cached interval of the highest precision yet calculated.

In most cases, computations that need to compare two algebraic numbers compute them with 128-bit precision intervals; if this does not suffice to prove that the numbers are different, then we fall back on exact computation.

Note that division involves an implicit comparison of the divisor against zero, and may thus trigger exact computation.

Also, using an algebraic number in the leading coefficient of a polynomial also involves an implicit comparison against zero, which again may trigger exact computation.

Note that we work fairly hard to avoid computing new number fields; to help, we keep a lattice of already-computed number fields and their inclusions.

EXAMPLES:

```
sage: sqrt(AA(2)) > 0
True
sage: (sqrt(5 + 2*sqrt(QQbar(6))) - sqrt(QQbar(3)))^2 == 2
True
sage: AA((sqrt(5 + 2*sqrt(6)) - sqrt(3))^2) == 2
True
```

For a monic cubic polynomial $x^3 + bx^2 + cx + d$ with roots $s1$, $s2$, $s3$, the discriminant is defined as $(s1 - s2)^2(s1 - s3)^2(s2 - s3)^2$ and can be computed as $b^2c^2 - 4b^3d - 4c^3 + 18bcd - 27d^2$. We can test that these definitions do give the same result:

```
sage: def disc1(b, c, d):
...       return b^2*c^2 - 4*b^3*d - 4*c^3 + 18*b*c*d - 27*d^2
sage: def disc2(s1, s2, s3):
...       return ((s1-s2)*(s1-s3)*(s2-s3))^2
sage: x = polygen(AA)
sage: p = x*(x-2)*(x-4)
sage: cp = AA.common_polynomial(p)
sage: d, c, b, _ = p.list()
sage: s1 = AA.polynomial_root(cp, RIF(-1, 1))
sage: s2 = AA.polynomial_root(cp, RIF(1, 3))
sage: s3 = AA.polynomial_root(cp, RIF(3, 5))
sage: disc1(b, c, d) == disc2(s1, s2, s3)
True
sage: p = p + 1
sage: cp = AA.common_polynomial(p)
sage: d, c, b, _ = p.list()
sage: s1 = AA.polynomial_root(cp, RIF(-1, 1))
sage: s2 = AA.polynomial_root(cp, RIF(1, 3))
sage: s3 = AA.polynomial_root(cp, RIF(3, 5))
sage: disc1(b, c, d) == disc2(s1, s2, s3)
True
sage: p = (x-sqrt(AA(2)))*(x-AA(2).nth_root(3))*(x-sqrt(AA(3)))
sage: cp = AA.common_polynomial(p)
sage: d, c, b, _ = p.list()
sage: s1 = AA.polynomial_root(cp, RIF(1.4, 1.5))
sage: s2 = AA.polynomial_root(cp, RIF(1.7, 1.8))
sage: s3 = AA.polynomial_root(cp, RIF(1.2, 1.3))
sage: disc1(b, c, d) == disc2(s1, s2, s3)
True
```

We can coerce from symbolic expressions:

```
sage: QQbar(sqrt(-5))
2.236067977499790?*I
sage: AA(sqrt(2) + sqrt(3))
3.146264369941973?
sage: QQbar(sqrt(2)) + sqrt(3)
3.146264369941973?
sage: sqrt(2) + QQbar(sqrt(3))
3.146264369941973?
sage: QQbar(I)
1*I
sage: AA(I)
```

```
Traceback (most recent call last):
...
TypeError: Illegal initializer for algebraic number
sage: QQbar(I * golden_ratio)
1.618033988749895?*I
sage: AA(golden_ratio)^2 - AA(golden_ratio)
1
sage: QQbar((-8)^(1/3))
1.000000000000000? + 1.732050807568878?*I
sage: AA((-8)^(1/3))
-2
sage: QQbar((-4)^(1/4))
1 + 1*I
sage: AA((-4)^(1/4))
Traceback (most recent call last):
...
ValueError: Cannot coerce algebraic number with non-zero imaginary part to algebraic real
```

Note the different behavior in taking roots: for `AA` we prefer real roots if they exist, but for `QQbar` we take the principal root:

```
sage: AA(-1)^(1/3)
-1
sage: QQbar(-1)^(1/3)
0.500000000000000? + 0.866025403784439?*I
```

We can explicitly coerce from $\mathbf{Q}[I]$. (Technically, this is not quite kosher, since $\mathbf{Q}[I]$ doesn't come with an embedding; we do not know whether the field generator is supposed to map to $+I$ or $-I$. We assume that for any quadratic field with polynomial $x^2 + 1$, the generator maps to $+I$.):

```
sage: K.<im> = QQ[I]
sage: pythag = QQbar(3/5 + 4*im/5); pythag
4/5*I + 3/5
sage: pythag.abs() == 1
True
```

However, implicit coercion from $\mathbf{Q}[I]$ is not allowed:

```
sage: QQbar(1) + im
Traceback (most recent call last):
...
TypeError: unsupported operand parent(s) for '+': 'Algebraic Field' and 'Number Field in I with defi
```

We can implicitly coerce from algebraic reals to algebraic numbers:

```
sage: a = QQbar(1); print a, a.parent()
1 Algebraic Field
sage: b = AA(1); print b, b.parent()
1 Algebraic Real Field
sage: c = a + b; print c, c.parent()
2 Algebraic Field
```

Some computation with radicals:

```
sage: phi = (1 + sqrt(AA(5))) / 2
sage: phi^2 == phi + 1
True
sage: tau = (1 - sqrt(AA(5))) / 2
```

```
sage: tau^2 == tau + 1
True
sage: phi + tau == 1
True
sage: tau < 0
True

sage: rt23 = sqrt(AA(2/3))
sage: rt35 = sqrt(AA(3/5))
sage: rt25 = sqrt(AA(2/5))
sage: rt23 * rt35 == rt25
True
```

The Sage rings `AA` and `QQbar` can decide equalities between radical expressions (over the reals and complex numbers respectively):

```
sage: a = AA((2/(3*sqrt(3)) + 10/27)^(1/3) - 2/(9*(2/(3*sqrt(3)) + 10/27)^(1/3)) + 1/3)
sage: a
1.000000000000000?
sage: a == 1
True
```

Algebraic numbers which are known to be rational print as rationals; otherwise they print as intervals (with 53-bit precision):

```
sage: AA(2)/3
2/3
sage: QQbar(5/7)
5/7
sage: QQbar(1/3 - 1/4*I)
-1/4*I + 1/3
sage: two = QQbar(4).nth_root(4)^2; two
2.000000000000000?
sage: two == 2; two
True
2
sage: phi
1.618033988749895?
```

We can find the real and imaginary parts of an algebraic number (exactly):

```
sage: r = QQbar.polynomial_root(x^5 - x - 1, CIF(RIF(0.1, 0.2), RIF(1.0, 1.1))); r
0.1812324444698754? + 1.083954101317711?*I
sage: r.real()
0.1812324444698754?
sage: r.imag()
1.083954101317711?
sage: r.minpoly()
x^5 - x - 1
sage: r.real().minpoly()
x^10 + 3/16*x^6 + 11/32*x^5 - 1/64*x^2 + 1/128*x - 1/1024
sage: r.imag().minpoly()  # long time (10s on sage.math, 2013)
x^20 - 5/8*x^16 - 95/256*x^12 - 625/1024*x^10 - 5/512*x^8 - 1875/8192*x^6 + 25/4096*x^4 - 625/32768*x
```

We can find the absolute value and norm of an algebraic number exactly. (Note that we define the norm as the product of a number and its complex conjugate; this is the algebraic definition of norm, if we view `QQbar` as `AA[I]`.):

```
sage: R.<x> = QQ[]
sage: r = (x^3 + 8).roots(QQbar, multiplicities=False)[2]; r
1.000000000000000? + 1.732050807568878?*I
sage: r.abs() == 2
True
sage: r.norm() == 4
True
sage: (r+I).norm().minpoly()
x^2 - 10*x + 13
sage: r = AA.polynomial_root(x^2 - x - 1, RIF(-1, 0)); r
-0.618033988749895?
sage: r.abs().minpoly()
x^2 + x - 1
```

We can compute the multiplicative order of an algebraic number:

```
sage: QQbar(-1/2 + I*sqrt(3)/2).multiplicative_order()
3
sage: QQbar(-sqrt(3)/2 + I/2).multiplicative_order()
12
sage: QQbar.zeta(12345).multiplicative_order()
12345
```

Cyclotomic fields are very fast as long as we only multiply and divide:

```
sage: z3_3 = QQbar.zeta(3) * 3
sage: z4_4 = QQbar.zeta(4) * 4
sage: z5_5 = QQbar.zeta(5) * 5
sage: z6_6 = QQbar.zeta(6) * 6
sage: z20_20 = QQbar.zeta(20) * 20
sage: z3_3 * z4_4 * z5_5 * z6_6 * z20_20
7200
```

And they are still pretty fast even if you add and subtract, and trigger exact computation:

```
sage: (z3_3 + z4_4 + z5_5 + z6_6 + z20_20)._exact_value()
4*zeta60^15 + 5*zeta60^12 + 9*zeta60^10 + 20*zeta60^3 - 3 where a^16 + a^14 - a^10 - a^8 - a^6 + a^2
```

The paper "ARPREC: An Arbitrary Precision Computation Package" by Bailey, Yozo, Li and Thompson discusses this result. Evidently it is difficult to find, but we can easily verify it.

```
sage: alpha = QQbar.polynomial_root(x^10 + x^9 - x^7 - x^6 - x^5 - x^4 - x^3 + x + 1, RIF(1, 1.2))
sage: lhs = alpha^630 - 1
sage: rhs_num = (alpha^315 - 1) * (alpha^210 - 1) * (alpha^126 - 1)^2 * (alpha^90 - 1) * (alpha^3 - 1
sage: rhs_den = (alpha^35 - 1) * (alpha^15 - 1)^2 * (alpha^14 - 1)^2 * (alpha^5 - 1)^6 * alpha^68
sage: rhs = rhs_num / rhs_den
sage: lhs
2.642040335819351?e44
sage: rhs
2.642040335819351?e44
sage: lhs - rhs
0.?e29
sage: lhs == rhs
True
sage: lhs - rhs
0
sage: lhs._exact_value()
106486994025108862293341329896296060022238 31*a^9 + 231745602491002861337181837128025290 35435800*a^8 -
```

Given an algebraic number, we can produce a string that will reproduce that algebraic number if you type the string into Sage. We can see that until exact computation is triggered, an algebraic number keeps track of the computation steps used to produce that number:

```
sage: rt2 = AA(sqrt(2))
sage: rt3 = AA(sqrt(3))
sage: n = (rt2 + rt3)^5; n
308.30180017229752?
sage: sage_input(n)
v1 = sqrt(AA(2)) + sqrt(AA(3))
v2 = v1*v1
v2*v2*v1
```

But once exact computation is triggered, the computation tree is discarded, and we get a way to produce the number directly:

```
sage: n == 109*rt2 + 89*rt3
True
sage: sage_input(n)
R.<x> = AA[]
v = AA.polynomial_root(AA.common_polynomial(x^4 - 4*x^2 + 1), RIF(RR(0.51763809020504148), RR(0.5176
-109*v^3 - 89*v^2 + 327*v + 178
```

We can also see that some computations (basically, those which are easy to perform exactly) are performed directly, instead of storing the computation tree:

```
sage: z3_3 = QQbar.zeta(3) * 3
sage: z4_4 = QQbar.zeta(4) * 4
sage: z5_5 = QQbar.zeta(5) * 5
sage: sage_input(z3_3 * z4_4 * z5_5)
-60*QQbar.zeta(60)^17
```

Note that the `verify=True` argument to `sage_input` will always trigger exact computation, so running `sage_input` twice in a row on the same number will actually give different answers. In the following, running `sage_input` on `n` will also trigger exact computation on `rt2`, as you can see by the fact that the third output is different than the first:

```
sage: rt2 = AA(sqrt(2))
sage: n = rt2^2
sage: sage_input(n, verify=True)
# Verified
v = sqrt(AA(2))
v*v
sage: sage_input(n, verify=True)
# Verified
AA(2)
sage: n = rt2^2
sage: sage_input(n, verify=True)
# Verified
AA(2)
```

Just for fun, let's try `sage_input` on a very complicated expression. The output of this example changed with the rewritting of polynomial multiplication algorithms in #10255:

```
sage: rt2 = sqrt(AA(2))
sage: rt3 = sqrt(QQbar(3))
sage: x = polygen(QQbar)
sage: nrt3 = AA.polynomial_root((x-rt2)*(x+rt3), RIF(-2, -1))
```

```
sage: one = AA.polynomial_root((x-rt2)*(x-rt3)*(x-nrt3)*(x-1-rt3-nrt3), RIF(0.9, 1.1))
sage: one
1.000000000000000?
sage: sage_input(one, verify=True)
# Verified
R.<x> = QQbar[]
v1 = AA(2)
v2 = QQbar(sqrt(v1))
v3 = QQbar(3)
v4 = sqrt(v3)
v5 = -v2 - v4
v6 = QQbar(sqrt(v1))
v7 = sqrt(v3)
cp = AA.common_polynomial(x^2 + (-v6 + v7)*x - v6*v7)
v8 = QQbar.polynomial_root(cp, RIF(-RR(1.7320508075688774), -RR(1.7320508075688772)))
v9 = v5 - v8
v10 = -1 - v4 - QQbar.polynomial_root(cp, RIF(-RR(1.7320508075688774), -RR(1.7320508075688772)))
v11 = v2*v4
v12 = v11 - v5*v8
si = v11*v8
AA.polynomial_root(AA.common_polynomial(x^4 + (v9 + v10)*x^3 + (v12 + v9*v10)*x^2 + (-si + v12*v10)*x
sage: one
1
```

We can pickle and unpickle algebraic fields (and they are globally unique):

```
sage: loads(dumps(AlgebraicField())) is AlgebraicField()
True
sage: loads(dumps(AlgebraicRealField())) is AlgebraicRealField()
True
```

We can pickle and unpickle algebraic numbers:

```
sage: loads(dumps(QQbar(10))) == QQbar(10)
True
sage: loads(dumps(QQbar(5/2))) == QQbar(5/2)
True
sage: loads(dumps(QQbar.zeta(5))) == QQbar.zeta(5)
True

sage: t = QQbar(sqrt(2)); type(t._descr)
<class 'sage.rings.qqbar.ANRoot'>
sage: loads(dumps(t)) == QQbar(sqrt(2))
True

sage: t.exactify(); type(t._descr)
<class 'sage.rings.qqbar.ANExtensionElement'>
sage: loads(dumps(t)) == QQbar(sqrt(2))
True

sage: t = ~QQbar(sqrt(2)); type(t._descr)
<class 'sage.rings.qqbar.ANUnaryExpr'>
sage: loads(dumps(t)) == 1/QQbar(sqrt(2))
True

sage: t = QQbar(sqrt(2)) + QQbar(sqrt(3)); type(t._descr)
<class 'sage.rings.qqbar.ANBinaryExpr'>
sage: loads(dumps(t)) == QQbar(sqrt(2)) + QQbar(sqrt(3))
```

```
True
```

We can convert elements of QQbar and AA into the following types: float, complex, RDF, CDF, RR, CC, RIF, CIF, ZZ, and QQ, with a few exceptions. (For the arbitrary-precision types, RR, CC, RIF, and CIF, it can convert into a field of arbitrary precision.)

Converting from QQbar to a real type (float, RDF, RR, RIF, ZZ, or QQ) succeeds only if the QQbar is actually real (has an imaginary component of exactly zero). Converting from either AA or QQbar to ZZ or QQ succeeds only if the number actually is an integer or rational. If conversion fails, a ValueError will be raised.

Here are examples of all of these conversions:

```
sage: all_vals = [AA(42), AA(22/7), AA(golden_ratio), QQbar(-13), QQbar(89/55), QQbar(-sqrt(7)), QQba
sage: def convert_test_all(ty):
....:     def convert_test(v):
....:         try:
....:             return ty(v)
....:         except ValueError:
....:             return None
....:     return [convert_test(_) for _ in all_vals]
sage: convert_test_all(float)
[42.0, 3.1428571428571432, 1.618033988749895, -13.0, 1.6181818181818182, -2.6457513110645907, None]
sage: convert_test_all(complex)
[(42+0j), (3.1428571428571432+0j), (1.618033988749895+0j), (-13+0j), (1.6181818181818182+0j), (-2.645
sage: convert_test_all(RDF)
[42.0, 3.1428571428571432, 1.618033988749895, -13.0, 1.6181818181818182, -2.6457513110645907, None]
sage: convert_test_all(CDF)
[42.0, 3.1428571428571432, 1.618033988749895, -13.0, 1.6181818181818182, -2.6457513110645907, 0.30901
sage: convert_test_all(RR)
[42.0000000000000, 3.14285714285714, 1.61803398874989, -13.0000000000000, 1.61818181818182, -2.645751
sage: convert_test_all(CC)
[42.0000000000000, 3.14285714285714, 1.61803398874989, -13.0000000000000, 1.61818181818182, -2.645751
sage: convert_test_all(RIF)
[42, 3.142857142857143?, 1.618033988749895?, -13, 1.618181818181819?, -2.645751311064591?, None]
sage: convert_test_all(CIF)
[42, 3.142857142857143?, 1.618033988749895?, -13, 1.618181818181819?, -2.645751311064591?, 0.3090169
sage: convert_test_all(ZZ)
[42, None, None, -13, None, None, None]
sage: convert_test_all(QQ)
[42, 22/7, None, -13, 89/55, None, None]
```

TESTS:

Verify that trac ticket #10981 is fixed:

```
sage: x = AA['x'].gen()
sage: P = 1/(1+x^4)
sage: P.partial_fraction_decomposition()
(0, [(-0.3535533905932738?*x + 1/2)/(x^2 - 1.414213562373095?*x + 1), (0.3535533905932738?*x + 1/2)/
```

class sage.rings.qqbar.**ANBinaryExpr**(*left*, *right*, *op*)

Bases: sage.rings.qqbar.ANDescr

Initialize this ANBinaryExpr.

EXAMPLE:

```
sage: t = QQbar(sqrt(2)) + QQbar(sqrt(3)); type(t._descr) # indirect doctest
<class 'sage.rings.qqbar.ANBinaryExpr'>
```

exactify()

TESTS:

```
sage: rt2c = QQbar.zeta(3) + AA(sqrt(2)) - QQbar.zeta(3)
sage: rt2c.exactify()
```

We check to make sure that this method still works even. We do this by increasing the recursion level at each step and decrease it before we return:

```
sage: import sys; sys.getrecursionlimit()
1000
sage: s = SymmetricFunctions(QQ).schur()
sage: a=s([3,2]).expand(8)(flatten([[QQbar.zeta(3)^d for d in range(3)], [QQbar.zeta(5)^d fo
sage: a.exactify(); a # long time
0
sage: sys.getrecursionlimit()
1000
```

handle_sage_input(*sib*, *coerce*, *is_qqbar*)

Produce an expression which will reproduce this value when evaluated, and an indication of whether this value is worth sharing (always True for `ANBinaryExpr`).

EXAMPLES:

```
sage: sage_input(2 + sqrt(AA(2)), verify=True)
# Verified
2 + sqrt(AA(2))
sage: sage_input(sqrt(AA(2)) + 2, verify=True)
# Verified
sqrt(AA(2)) + 2
sage: sage_input(2 - sqrt(AA(2)), verify=True)
# Verified
2 - sqrt(AA(2))
sage: sage_input(2 / sqrt(AA(2)), verify=True)
# Verified
2/sqrt(AA(2))
sage: sage_input(2 + (-1*sqrt(AA(2))), verify=True)
# Verified
2 - sqrt(AA(2))
sage: sage_input(2*sqrt(AA(2)), verify=True)
# Verified
2*sqrt(AA(2))
sage: rt2 = sqrt(AA(2))
sage: one = rt2/rt2
sage: n = one+3
sage: sage_input(n)
v = sqrt(AA(2))
v/v + 3
sage: one == 1
True
sage: sage_input(n)
1 + AA(3)
sage: rt3 = QQbar(sqrt(3))
sage: one = rt3/rt3
sage: n = sqrt(AA(2))+one
sage: one == 1
True
sage: sage_input(n)
QQbar(sqrt(AA(2))) + 1
sage: from sage.rings.qqbar import *
```

```
sage: from sage.misc.sage_input import SageInputBuilder
sage: sib = SageInputBuilder()
sage: binexp = ANBinaryExpr(AA(3), AA(5), '*')
sage: binexp.handle_sage_input(sib, False, False)
({binop:* {atomic:3} {call: {atomic:AA}({atomic:5})}}, True)
sage: binexp.handle_sage_input(sib, False, True)
({call: {atomic:QQbar}({binop:* {atomic:3} {call: {atomic:AA}({atomic:5})}})}, True)
```

is_complex()

Whether this element is complex. Does not trigger exact computation, so may return True even if the element is real.

EXAMPLE:

```
sage: x = (QQbar(sqrt(-2)) / QQbar(sqrt(-5)))._descr
sage: x.is_complex()
True
```

kind()

Return a string describing what kind of element this is. Returns 'other'.

EXAMPLE:

```
sage: x = (QQbar(sqrt(2)) + QQbar(sqrt(5)))._descr
sage: type(x)
<class 'sage.rings.qqbar.ANBinaryExpr'>
sage: x.kind()
'other'
```

class sage.rings.qqbar.**ANDescr**

Bases: sage.structure.sage_object.SageObject

An AlgebraicNumber or AlgebraicReal is a wrapper around an ANDescr object. ANDescr is an abstract base class, which should never be directly instantiated; its concrete subclasses are ANRational, ANBinaryExpr, ANUnaryExpr, ANRootOfUnity, ANRoot, and ANExtensionElement. ANDescr and all of its subclasses are for internal use, and should not be used directly.

abs(*n*)

Absolute value of self.

EXAMPLE:

```
sage: a = QQbar(sqrt(2))
sage: b = a._descr
sage: b.abs(a)
<class 'sage.rings.qqbar.ANUnaryExpr'>
```

conjugate(*n*)

Complex conjugate of self.

EXAMPLE:

```
sage: a = QQbar(sqrt(-7))
sage: b = a._descr
sage: b.conjugate(a)
<class 'sage.rings.qqbar.ANUnaryExpr'>
```

imag(*n*)

Imaginary part of self.

EXAMPLE:

```
sage: a = QQbar(sqrt(-7))
sage: b = a._descr
sage: b.imag(a)
<class 'sage.rings.qqbar.ANUnaryExpr'>
```

invert(*n*)

1/self.

EXAMPLE:

```
sage: a = QQbar(sqrt(2))
sage: b = a._descr
sage: b.invert(a)
<class 'sage.rings.qqbar.ANUnaryExpr'>
```

is_exact()

Returns True if self is an ANRational, ANRootOfUnity, or ANExtensionElement.

EXAMPLES:

```
sage: from sage.rings.qqbar import ANRational
sage: ANRational(1/2).is_exact()
True
sage: QQbar(3+I)._descr.is_exact()
True
sage: QQbar.zeta(17)._descr.is_exact()
True
```

is_field_element()

Returns `True` if self is an `ANExtensionElement`.

EXAMPLES:

```
sage: from sage.rings.qqbar import ANExtensionElement, ANRoot, AlgebraicGenerator
sage: _.<y> = QQ['y']
sage: x = polygen(QQbar)
sage: nf2 = NumberField(y^2 - 2, name='a', check=False)
sage: root2 = ANRoot(x^2 - 2, RIF(1, 2))
sage: gen2 = AlgebraicGenerator(nf2, root2)
sage: sqrt2 = ANExtensionElement(gen2, nf2.gen())
sage: sqrt2.is_field_element()
True
```

is_rational()

Returns `True` if self is an `ANRational` object. (Note that the constructors for `ANExtensionElement` and `ANRootOfUnity` will actually return `ANRational` objects for rational numbers.)

EXAMPLES:

```
sage: from sage.rings.qqbar import ANRational
sage: ANRational(3/7).is_rational()
True
```

is_simple()

Checks whether this descriptor represents a value with the same algebraic degree as the number field associated with the descriptor.

Returns `True` if self is an `ANRational`, `ANRootOfUnit`, or a minimal `ANExtensionElement`.

EXAMPLES:

```
sage: from sage.rings.qqbar import ANRational
sage: ANRational(1/2).is_simple()
True
sage: rt2 = AA(sqrt(2))
sage: rt3 = AA(sqrt(3))
sage: rt2b = rt3 + rt2 - rt3
sage: rt2.exactify()
sage: rt2._descr.is_simple()
True
sage: rt2b.exactify()
sage: rt2b._descr.is_simple()
False
sage: rt2b.simplify()
sage: rt2b._descr.is_simple()
True
```

neg(*n*)

Negation of self.

EXAMPLE:

```
sage: a = QQbar(sqrt(2))
sage: b = a._descr
sage: b.neg(a)
<class 'sage.rings.qqbar.ANUnaryExpr'>
```

norm(*n*)

Field norm of self from $\overline{\mathbf{Q}}$ to its real subfield $\mathbf{A}$, i.e.~the square of the usual complex absolute value.

EXAMPLE:

```
sage: a = QQbar(sqrt(-7))
sage: b = a._descr
sage: b.norm(a)
<class 'sage.rings.qqbar.ANUnaryExpr'>
```

real(*n*)

Real part of self.

EXAMPLE:

```
sage: a = QQbar(sqrt(-7))
sage: b = a._descr
sage: b.real(a)
<class 'sage.rings.qqbar.ANUnaryExpr'>
```

class sage.rings.qqbar.**ANExtensionElement**(*generator*, *value*)

Bases: sage.rings.qqbar.ANDescr

The subclass of ANDescr that represents a number field element in terms of a specific generator. Consists of a polynomial with rational coefficients in terms of the generator, and the generator itself, an AlgebraicGenerator.

abs(*n*)

Return the absolute value of self (square root of the norm).

EXAMPLE:

```
sage: a = QQbar(sqrt(-2)) + QQbar(sqrt(-3))
sage: a.exactify()
sage: b = a._descr
```

```
sage: type(b)
<class 'sage.rings.qqbar.ANExtensionElement'>
sage: b.abs(a)
Root 3.146264369941972342? of x^2 - 9.898979485566362
```

conjugate(*n*)

Negation of self.

EXAMPLE:

```
sage: a = QQbar(sqrt(-2)) + QQbar(sqrt(-3))
sage: a.exactify()
sage: b = a._descr
sage: type(b)
<class 'sage.rings.qqbar.ANExtensionElement'>
sage: b.conjugate(a)
-1/3*a^3 + 2/3*a^2 - 4/3*a + 2 where a^4 - 2*a^3 + a^2 - 6*a + 9 = 0 and a in -0.72474487139
sage: b.conjugate("ham spam and eggs")
-1/3*a^3 + 2/3*a^2 - 4/3*a + 2 where a^4 - 2*a^3 + a^2 - 6*a + 9 = 0 and a in -0.72474487139
```

exactify()

Return an exact representation of self. Since self is already exact, just return self.

EXAMPLE:

```
sage: v = (x^2 - x - 1).roots(ring=AA, multiplicities=False)[1]._descr.exactify()
sage: type(v)
<class 'sage.rings.qqbar.ANExtensionElement'>
sage: v.exactify() is v
True
```

field_element_value()

Return the underlying number field element.

EXAMPLE:

```
sage: v = (x^2 - x - 1).roots(ring=AA, multiplicities=False)[1]._descr.exactify()
sage: v.field_element_value()
a
```

gaussian_value()

Return self as an element of $\mathbf{Q}(i)$.

EXAMPLE:

```
sage: a = QQbar(I) + 3/7
sage: a.exactify()
sage: b = a._descr
sage: type(b)
<class 'sage.rings.qqbar.ANExtensionElement'>
sage: b.gaussian_value()
I + 3/7
```

A non-example:

```
sage: a = QQbar(sqrt(-2)) + QQbar(sqrt(-3))
sage: a.exactify()
sage: b = a._descr
sage: type(b)
<class 'sage.rings.qqbar.ANExtensionElement'>
sage: b.gaussian_value()
```

```
Traceback (most recent call last):
...
AssertionError
```

generator()

Return the AlgebraicGenerator object corresponding to self.

EXAMPLE:

```
sage: v = (x^2 - x - 1).roots(ring=AA, multiplicities=False)[1]._descr.exactify()
sage: v.generator()
Number Field in a with defining polynomial y^2 - y - 1 with a in 1.618033988749895?
```

handle_sage_input(*sib*, *coerce*, *is_qqbar*)

Produce an expression which will reproduce this value when evaluated, and an indication of whether this value is worth sharing (always True, for ANExtensionElement).

EXAMPLES:

```
sage: I = QQbar(I)
sage: sage_input(3+4*I, verify=True)
# Verified
QQbar(3 + 4*I)
sage: v = QQbar.zeta(3) + QQbar.zeta(5)
sage: v - v == 0
True
sage: sage_input(vector(QQbar, (4-3*I, QQbar.zeta(7))), verify=True)
# Verified
vector(QQbar, [4 - 3*I, QQbar.zeta(7)])
sage: sage_input(v, verify=True)
# Verified
v = QQbar.zeta(15)
v^5 + v^3
sage: v = QQbar(sqrt(AA(2)))
sage: v.exactify()
sage: sage_input(v, verify=True)
# Verified
R.<x> = AA[]
QQbar(AA.polynomial_root(AA.common_polynomial(x^2 - 2), RIF(RR(1.4142135623730949), RR(1.414
sage: from sage.rings.qqbar import *
sage: from sage.misc.sage_input import SageInputBuilder
sage: sib = SageInputBuilder()
sage: extel = ANExtensionElement(QQbar_I_generator, QQbar_I_generator.field().gen() + 1)
sage: extel.handle_sage_input(sib, False, True)
({call: {atomic:QQbar}({binop:+ {atomic:1} {atomic:I}})}, True)
```

invert(*n*)

1/self.

EXAMPLE:

```
sage: a = QQbar(sqrt(-2)) + QQbar(sqrt(-3))
sage: a.exactify()
sage: b = a._descr
sage: type(b)
<class 'sage.rings.qqbar.ANExtensionElement'>
sage: b.invert(a)
7/3*a^3 - 2/3*a^2 + 4/3*a - 12 where a^4 - 2*a^3 + a^2 - 6*a + 9 = 0 and a in -0.72474487139
sage: b.invert("ham spam and eggs")
7/3*a^3 - 2/3*a^2 + 4/3*a - 12 where a^4 - 2*a^3 + a^2 - 6*a + 9 = 0 and a in -0.72474487139
```

is_complex()

Return True if the number field that defines this element is not real. This does not imply that the element itself is definitely non-real, as in the example below.

EXAMPLE:

```
sage: rt2 = QQbar(sqrt(2))
sage: rtm3 = QQbar(sqrt(-3))
sage: x = rtm3 + rt2 - rtm3
sage: x.exactify()
sage: y = x._descr
sage: type(y)
<class 'sage.rings.qqbar.ANExtensionElement'>
sage: y.is_complex()
True
sage: x.imag() == 0
True
```

is_exact()

Return True, since ANExtensionElements are exact.

EXAMPLE:

```
sage: rt2 = QQbar(sqrt(2))
sage: rtm3 = QQbar(sqrt(-3))
sage: x = rtm3 + rt2 - rtm3
sage: x.exactify()
sage: y = x._descr
sage: type(y)
<class 'sage.rings.qqbar.ANExtensionElement'>
sage: y.is_exact()
True
```

is_field_element()

Return True if self is an element of a number field (always true for ANExtensionElements)

EXAMPLE:

```
sage: v = (x^2 - x - 1).roots(ring=AA, multiplicities=False)[1]._descr.exactify()
sage: v.is_field_element()
True
```

is_simple()

Checks whether this descriptor represents a value with the same algebraic degree as the number field associated with the descriptor.

For `ANExtensionElement` elements, we check this by comparing the degree of the minimal polynomial to the degree of the field.

EXAMPLES:

```
sage: rt2 = AA(sqrt(2))
sage: rt3 = AA(sqrt(3))
sage: rt2b = rt3 + rt2 - rt3
sage: rt2.exactify()
sage: rt2._descr
a where a^2 - 2 = 0 and a in 1.414213562373095?
sage: rt2._descr.is_simple()
True
```

```
sage: rt2b.exactify()
sage: rt2b._descr
a^3 - 3*a where a^4 - 4*a^2 + 1 = 0 and a in 1.931851652578137?
sage: rt2b._descr.is_simple()
False
```

kind()

Return a string describing what kind of element this is.

EXAMPLE:

```
sage: x = QQbar(sqrt(2) + sqrt(3))
sage: x.exactify()
sage: x._descr.kind()
'element'
sage: x = QQbar(I) + 1
sage: x.exactify()
sage: x._descr.kind()
'gaussian'
```

minpoly()

Compute the minimal polynomial of this algebraic number.

EXAMPLES:

```
sage: v = (x^2 - x - 1).roots(ring=AA, multiplicities=False)[1]._descr.exactify()
sage: type(v)
<class 'sage.rings.qqbar.ANExtensionElement'>
sage: v.minpoly()
x^2 - x - 1
```

neg(*n*)

Negation of self.

EXAMPLE:

```
sage: a = QQbar(sqrt(-2)) + QQbar(sqrt(-3))
sage: a.exactify()
sage: b = a._descr
sage: type(b)
<class 'sage.rings.qqbar.ANExtensionElement'>
sage: b.neg(a)
1/3*a^3 - 2/3*a^2 + 4/3*a - 2 where a^4 - 2*a^3 + a^2 - 6*a + 9 = 0 and a in -0.724744871391
sage: b.neg("ham spam and eggs")
1/3*a^3 - 2/3*a^2 + 4/3*a - 2 where a^4 - 2*a^3 + a^2 - 6*a + 9 = 0 and a in -0.724744871391
```

norm(*n*)

Norm of self (square of complex absolute value)

EXAMPLE:

```
sage: a = QQbar(sqrt(-2)) + QQbar(sqrt(-3))
sage: a.exactify()
sage: b = a._descr
sage: type(b)
<class 'sage.rings.qqbar.ANExtensionElement'>
sage: b.norm(a)
<class 'sage.rings.qqbar.ANUnaryExpr'>
```

rational_argument(*n*)

If the argument of self is 2π times some rational number in $[1/2, -1/2)$, return that rational; otherwise, return None.

EXAMPLE:

```
sage: a = QQbar(sqrt(-2)) + QQbar(sqrt(3))
sage: a.exactify()
sage: b = a._descr
sage: type(b)
<class 'sage.rings.qqbar.ANExtensionElement'>
sage: b.rational_argument(a) is None
True
sage: x = polygen(QQ)
sage: a = (x^4 + 1).roots(QQbar, multiplicities=False)[0]
sage: a.exactify()
sage: b = a._descr
sage: b.rational_argument(a)
-3/8
```

simplify(*n*)

Compute an exact representation for this descriptor, in the smallest possible number field.

INPUT:

- n – The element of AA or QQbar corresponding to this descriptor.

EXAMPLES:

```
sage: rt2 = AA(sqrt(2))
sage: rt3 = AA(sqrt(3))
sage: rt2b = rt3 + rt2 - rt3
sage: rt2b.exactify()
sage: rt2b._descr
a^3 - 3*a where a^4 - 4*a^2 + 1 = 0 and a in 1.931851652578137?
sage: rt2b._descr.simplify(rt2b)
a where a^2 - 2 = 0 and a in 1.414213562373095?
```

class sage.rings.qqbar.**ANRational**(*x*)

Bases: sage.rings.qqbar.ANDescr

The subclass of ANDescr that represents an arbitrary rational. This class is private, and should not be used directly.

abs(*n*)

Absolute value of self.

EXAMPLE:

```
sage: a = QQbar(3)
sage: b = a._descr
sage: b.abs(a)
3
```

angle()

Return a rational number $q \in (-1/2, 1/2]$ such that self is a rational multiple of $e^{2\pi i q}$. Always returns 0, since this element is rational.

EXAMPLE:

```
sage: QQbar(3)._descr.angle()
0
sage: QQbar(-3)._descr.angle()
0
```

```
sage: QQbar(0)._descr.angle()
0
```

exactify()
Calculate self exactly. Since self is a rational number, return self.

EXAMPLE:

```
sage: a = QQbar(1/3)._descr
sage: a.exactify() is a
True
```

gaussian_value()
Return self as an element of $\mathbf{Q}(i)$.

EXAMPLE:

```
sage: a = QQbar(3)
sage: b = a._descr
sage: x = b.gaussian_value(); x
3
sage: x.parent()
Number Field in I with defining polynomial x^2 + 1
```

generator()
Return an AlgebraicGenerator object associated to this element. Returns the trivial generator, since self is rational.

EXAMPLE:

```
sage: QQbar(0)._descr.generator()
Trivial generator
```

handle_sage_input(*sib*, *coerce*, *is_qqbar*)
Produce an expression which will reproduce this value when evaluated, and an indication of whether this value is worth sharing (always False, for rationals).

EXAMPLES:

```
sage: sage_input(QQbar(22/7), verify=True)
# Verified
QQbar(22/7)
sage: sage_input(-AA(3)/5, verify=True)
# Verified
AA(-3/5)
sage: sage_input(vector(AA, (0, 1/2, 1/3)), verify=True)
# Verified
vector(AA, [0, 1/2, 1/3])
sage: from sage.rings.qqbar import *
sage: from sage.misc.sage_input import SageInputBuilder
sage: sib = SageInputBuilder()
sage: rat = ANRational(9/10)
sage: rat.handle_sage_input(sib, False, True)
({call: {atomic:QQbar}({binop:/ {atomic:9} {atomic:10}})}, False)
```

invert(*n*)
1/self.

EXAMPLE:

```
sage: a = QQbar(3)
sage: b = a._descr
sage: b.invert(a)
1/3
```

is_complex()
Return False, since rational numbers are real

EXAMPLE:

```
sage: QQbar(1/7)._descr.is_complex()
False
```

is_exact()
Return True, since rationals are exact.

EXAMPLE:

```
sage: QQbar(1/3)._descr.is_exact()
True
```

is_rational()
Return True, since this is a rational number.

EXAMPLE:

```
sage: QQbar(34/9)._descr.is_rational()
True
sage: QQbar(0)._descr.is_rational()
True
```

is_simple()
Checks whether this descriptor represents a value with the same algebraic degree as the number field associated with the descriptor.

This is always true for rational numbers.

EXAMPLES:

```
sage: AA(1/2)._descr.is_simple()
True
```

kind()
Return a string describing what kind of element this is. Since this is a rational number, the result is either `'zero'` or `'rational'`.

EXAMPLES:

```
sage: a = QQbar(3)._descr; type(a)
<class 'sage.rings.qqbar.ANRational'>
sage: a.kind()
'rational'
sage: a = QQbar(0)._descr; type(a)
<class 'sage.rings.qqbar.ANRational'>
sage: a.kind()
'zero'
```

minpoly()
Return the min poly of self over $\mathbf{Q}$.

EXAMPLE:

```
sage: QQbar(7)._descr.minpoly()
x - 7
```

neg(*n*)
Negation of self.

EXAMPLE:

```
sage: a = QQbar(3)
sage: b = a._descr
sage: type(b)
<class 'sage.rings.qqbar.ANRational'>
sage: b.neg(a)
-3
```

rational_argument(*n*)
Return the argument of self divided by 2π, or `None` if this element is 0.

EXAMPLE:

```
sage: QQbar(3)._descr.rational_argument(None)
0
sage: QQbar(-3)._descr.rational_argument(None)
1/2
sage: QQbar(0)._descr.rational_argument(None) is None
True
```

rational_value()
Return self as a rational number.

EXAMPLE:

```
sage: a = QQbar(789/19)
sage: b = a._descr.rational_value(); b
789/19
sage: type(b)
<type 'sage.rings.rational.Rational'>
```

scale()
Return a rational number r such that `self` is equal to $re^{2\pi iq}$ for some $q \in (-1/2, 1/2]$. In other words, just return self as a rational number.

EXAMPLE:

```
sage: QQbar(-3)._descr.scale()
-3
```

class sage.rings.qqbar.**ANRoot**(*poly*, *interval*, *multiplicity=1*, *is_pow=None*)
Bases: sage.rings.qqbar.ANDescr

The subclass of `ANDescr` that represents a particular root of a polynomial with algebraic coefficients. This class is private, and should not be used directly.

conjugate(*n*)
Complex conjugate of this ANRoot object.

EXAMPLE:

```
sage: a = (x^2 + 23).roots(ring=QQbar, multiplicities=False)[0]
sage: b = a._descr
sage: type(b)
<class 'sage.rings.qqbar.ANRoot'>
```

```
sage: c = b.conjugate(a); c
<class 'sage.rings.qqbar.ANUnaryExpr'>
sage: c.exactify()
-2*a + 1 where a^2 - a + 6 = 0 and a in 0.50000000000000000? - 2.397915761656360?*I
```

exactify()

Returns either an ANRational or an ANExtensionElement with the same value as this number.

EXAMPLES:

```
sage: from sage.rings.qqbar import ANRoot
sage: x = polygen(QQbar)
sage: two = ANRoot((x-2)*(x-sqrt(QQbar(2))), RIF(1.9, 2.1))
sage: two.exactify()
2
sage: two.exactify().rational_value()
2
sage: strange = ANRoot(x^2 + sqrt(QQbar(3))*x - sqrt(QQbar(2)), RIF(-0, 1))
sage: strange.exactify()
a where a^8 - 6*a^6 + 5*a^4 - 12*a^2 + 4 = 0 and a in 0.6051012265139511?
```

TESTS:

Verify that trac ticket #12727 is fixed:

```
sage: m = sqrt(sin(pi/5)); a = QQbar(m); b = AA(m)
sage: a.minpoly()
x^8 - 5/4*x^4 + 5/16
sage: b.minpoly()
x^8 - 5/4*x^4 + 5/16
```

handle_sage_input(*sib*, *coerce*, *is_qqbar*)

Produce an expression which will reproduce this value when evaluated, and an indication of whether this value is worth sharing (always True, for ANRoot).

EXAMPLES:

```
sage: sage_input((AA(3)^(1/2))^(1/3), verify=True)
# Verified
sqrt(AA(3)).nth_root(3)
```

These two examples are too big to verify quickly. (Verification would create a field of degree 28.):

```
sage: sage_input((sqrt(AA(3))^(5/7))^(9/4))
(sqrt(AA(3))^(5/7))^(9/4)
sage: sage_input((sqrt(QQbar(-7))^(5/7))^(9/4))
(sqrt(QQbar(-7))^(5/7))^(9/4)
sage: x = polygen(QQ)
sage: sage_input(AA.polynomial_root(x^2-x-1, RIF(1, 2)), verify=True)
# Verified
R.<x> = AA[]
AA.polynomial_root(AA.common_polynomial(x^2 - x - 1), RIF(RR(1.6180339887498947), RR(1.61803
sage: sage_input(QQbar.polynomial_root(x^3-5, CIF(RIF(-3, 0), RIF(0, 3))), verify=True)
# Verified
R.<x> = AA[]
QQbar.polynomial_root(AA.common_polynomial(x^3 - 5), CIF(RIF(-RR(0.85498797333834853), -RR(0
sage: from sage.rings.qqbar import *
sage: from sage.misc.sage_input import SageInputBuilder
sage: sib = SageInputBuilder()
sage: rt = ANRoot(x^3 - 2, RIF(0, 4))
```

```
sage: rt.handle_sage_input(sib, False, True)
({call: {getattr: {atomic:QQbar}.polynomial_root}({call: {getattr: {atomic:AA}.common_polync
```

is_complex()

Whether this is a root in $\overline{\mathbf{Q}}$ (rather than $\mathbf{A}$). Note that this may return True even if the root is actually real, as the second example shows; it does *not* trigger exact computation to see if the root is real.

EXAMPLE:

```
sage: x = polygen(QQ)
sage: (x^2 - x - 1).roots(ring=AA, multiplicities=False)[1]._descr.is_complex()
False
sage: (x^2 - x - 1).roots(ring=QQbar, multiplicities=False)[1]._descr.is_complex()
True
```

kind()

Return a string indicating what kind of element this is.

EXAMPLE:

```
sage: (x^2 - x - 1).roots(ring=AA, multiplicities=False)[1]._descr.kind()
'other'
```

refine_interval(*interval*, *prec*)

Takes an interval which is assumed to enclose exactly one root of the polynomial (or, with multiplicity='k', exactly one root of the $k-1$-st derivative); and a precision, in bits.

Tries to find a narrow interval enclosing the root using interval arithmetic of the given precision. (No particular number of resulting bits of precision is guaranteed.)

Uses a combination of Newton's method (adapted for interval arithmetic) and bisection. The algorithm will converge very quickly if started with a sufficiently narrow interval.

EXAMPLES:

```
sage: from sage.rings.qqbar import ANRoot
sage: x = polygen(AA)
sage: rt2 = ANRoot(x^2 - 2, RIF(0, 2))
sage: rt2.refine_interval(RIF(0, 2), 75)
1.41421356237309504880?
```

class sage.rings.qqbar.**ANRootOfUnity**(*angle*, *scale*)

Bases: sage.rings.qqbar.ANDescr

The subclass of ANDescr that represents a rational multiplied by a root of unity. This class is private, and should not be used directly.

Such numbers are represented by a "rational angle" and a rational scale. The "rational angle" is the argument of the number, divided by 2π; so given angle α and scale s, the number is: $s(\cos(2\pi\alpha) + \sin(2\pi\alpha)i)$; or equivalently $s(e^{2\pi\alpha i})$.

We normalize so that $0 < \alpha < \frac{1}{2}$; this requires allowing both positive and negative scales. (Attempts to create an ANRootOfUnity with an angle which is a multiple of $\frac{1}{2}$ end up creating an ANRational instead.)

abs(*n*)

Absolute value of self.

EXAMPLE:

```
sage: a = -QQbar.zeta(17)^5 * 4/3; a._descr
-4/3*e^(2*pi*I*5/17)
```

```
sage: a._descr.abs(None)
4/3
```

angle()

Return the rational $\theta \in [0, 1/2)$ such that self represents a rational multiple of $e^{2\pi i\theta}$.

EXAMPLE:

```
sage: (-QQbar.zeta(3))._descr.angle()
1/3
sage: (-QQbar.zeta(3))._descr.rational_argument(None)
-1/6
```

conjugate(n)

Complex conjugate of self.

EXAMPLE:

```
sage: a = QQbar.zeta(17)^5 * 4/3; a._descr
4/3*e^(2*pi*I*5/17)
sage: a._descr.conjugate(None)
-4/3*e^(2*pi*I*7/34)
```

exactify()

Return self, since ANRootOfUnity elements are exact.

EXAMPLE:

```
sage: t = (QQbar.zeta(17)^13)._descr
sage: type(t)
<class 'sage.rings.qqbar.ANRootOfUnity'>
sage: t.exactify() is t
True
```

field_element_value()

Return self as an element of a cyclotomic field.

EXAMPLE:

```
sage: t = (QQbar.zeta(17)^13)._descr
sage: type(t)
<class 'sage.rings.qqbar.ANRootOfUnity'>
sage: s = t.field_element_value(); s
-zeta34^9
sage: s.parent()
Cyclotomic Field of order 34 and degree 16
```

gaussian_value()

Return self as an element of $\mathbf{Q}(i)$‘ (assuming this is possible).

EXAMPLE:

```
sage: (-17*QQbar.zeta(4))._descr.gaussian_value()
-17*I
sage: (-17*QQbar.zeta(5))._descr.gaussian_value()
Traceback (most recent call last):
...
AssertionError
```

generator()

Return an AlgebraicGenerator object corresponding to this element.

EXAMPLE:

```
sage: t = (QQbar.zeta(17)^13)._descr
sage: type(t)
<class 'sage.rings.qqbar.ANRootOfUnity'>
sage: t.generator()
1*e^(2*pi*I*1/34)
```

handle_sage_input(*sib*, *coerce*, *is_qqbar*)

Produce an expression which will reproduce this value when evaluated, and an indication of whether this value is worth sharing (False for imaginary numbers, True for others).

EXAMPLES:

```
sage: sage_input(22/7*QQbar.zeta(4), verify=True)
# Verified
QQbar(22/7*I)
sage: sage_input((2*QQbar.zeta(12))^4, verify=True)
# Verified
16*QQbar.zeta(3)
sage: sage_input(QQbar.zeta(5)^2, verify=True)
# Verified
QQbar.zeta(5)^2
sage: sage_input(QQbar.zeta(5)^3, verify=True)
# Verified
-QQbar.zeta(10)
sage: sage_input(vector(QQbar, (I, 3*QQbar.zeta(9))), verify=True)
# Verified
vector(QQbar, [I, 3*QQbar.zeta(9)])
sage: from sage.rings.qqbar import *
sage: from sage.misc.sage_input import SageInputBuilder
sage: sib = SageInputBuilder()
sage: rtofunity = ANRootOfUnity(137/500, 1/1000)
sage: rtofunity.handle_sage_input(sib, False, True)
({binop:* {binop:/ {atomic:1} {atomic:1000}} {binop:** {call: {getattr: {atomic:QQbar}.zeta}
```

invert(*n*)

1/self.

EXAMPLE:

```
sage: a = QQbar.zeta(17)^5 * 4/3; a._descr
4/3*e^(2*pi*I*5/17)
sage: a._descr.invert(None)
-3/4*e^(2*pi*I*7/34)
```

is_complex()

Return True, since this class is only used for complex algebraic numbers.

EXAMPLE:

```
sage: t = (QQbar.zeta(17)^13)._descr
sage: type(t)
<class 'sage.rings.qqbar.ANRootOfUnity'>
sage: t.is_complex()
True
```

is_exact()

Return True, since `ANRootOfUnity` elements are exact.

EXAMPLE:

```
sage: t = (QQbar.zeta(17)^13)._descr
sage: type(t)
<class 'sage.rings.qqbar.ANRootOfUnity'>
sage: t.is_exact()
True
```

is_simple()

Checks whether this descriptor represents a value with the same algebraic degree as the number field associated with the descriptor.

This is always true for `ANRootOfUnity` elements.

EXAMPLES:

```
sage: a = QQbar.zeta(17)^5 * 4/3; a._descr
4/3*e^(2*pi*I*5/17)
sage: a._descr.is_simple()
True
```

kind()

Return a string describing what kind of element this is.

EXAMPLE:

```
sage: QQbar.zeta(4)._descr.kind()
'imaginary'
sage: QQbar.zeta(5)._descr.kind()
'rootunity'
```

minpoly()

EXAMPLES:

```
sage: a = QQbar.zeta(7) * 2; a
1.246979603717467? + 1.563662964936060?*I
sage: a.minpoly()
x^6 + 2*x^5 + 4*x^4 + 8*x^3 + 16*x^2 + 32*x + 64
sage: a.minpoly()(a)
0.?e-15 + 0.?e-15*I
sage: a.minpoly()(a) == 0
True
```

neg(*n*)

Negation of self.

EXAMPLE:

```
sage: a = QQbar.zeta(17)^5 * 4/3; a._descr
4/3*e^(2*pi*I*5/17)
sage: a._descr.neg(None)
-4/3*e^(2*pi*I*5/17)
```

norm(*n*)

Norm (square of absolute value) of self.

EXAMPLE:

```
sage: a = -QQbar.zeta(17)^5 * 4/3; a._descr
-4/3*e^(2*pi*I*5/17)
sage: a._descr.norm(None)
16/9
```

rational_argument(*n*)
Return the rational $\theta \in (-1/2, 1/2)$ such that self represents a positive rational multiple of $e^{2\pi i\theta}$.

EXAMPLE:

```
sage: (-QQbar.zeta(3))._descr.angle()
1/3
sage: (-QQbar.zeta(3))._descr.rational_argument(None)
-1/6
```

scale()
Return the scale of self, the unique rational r such that self is equal to $re^{2\pi i\theta}$ for some $theta \in (-1/2, 1/2]$. This is ± 1 times `self.abs()`.

EXAMPLE:

```
sage: (QQbar.zeta(5)^3)._descr.scale()
-1
```

class sage.rings.qqbar.**ANUnaryExpr**(*arg*, *op*)
Bases: sage.rings.qqbar.ANDescr

Initialize this ANUnaryExpr.

EXAMPLE:

```
sage: t = ~QQbar(sqrt(2)); type(t._descr) # indirect doctest
<class 'sage.rings.qqbar.ANUnaryExpr'>
```

exactify()
Trigger exact computation of self.

EXAMPLE:

```
sage: v = (-QQbar(sqrt(2)))._descr
sage: type(v)
<class 'sage.rings.qqbar.ANUnaryExpr'>
sage: v.exactify()
-a where a^2 - 2 = 0 and a in 1.414213562373095?
```

handle_sage_input(*sib*, *coerce*, *is_qqbar*)
Produce an expression which will reproduce this value when evaluated, and an indication of whether this value is worth sharing (always True for `ANUnaryExpr`).

EXAMPLES:

```
sage: sage_input(-sqrt(AA(2)), verify=True)
# Verified
-sqrt(AA(2))
sage: sage_input(~sqrt(AA(2)), verify=True)
# Verified
~sqrt(AA(2))
sage: sage_input(sqrt(QQbar(-3)).conjugate(), verify=True)
# Verified
sqrt(QQbar(-3)).conjugate()
sage: sage_input(QQbar.zeta(3).real(), verify=True)
# Verified
QQbar.zeta(3).real()
sage: sage_input(QQbar.zeta(3).imag(), verify=True)
# Verified
QQbar.zeta(3).imag()
sage: sage_input(abs(sqrt(QQbar(-3))), verify=True)
```

```
# Verified
abs(sqrt(QQbar(-3)))
sage: sage_input(sqrt(QQbar(-3)).norm(), verify=True)
# Verified
sqrt(QQbar(-3)).norm()
sage: sage_input(QQbar(QQbar.zeta(3).real()), verify=True)
# Verified
QQbar(QQbar.zeta(3).real())
sage: from sage.rings.qqbar import *
sage: from sage.misc.sage_input import SageInputBuilder
sage: sib = SageInputBuilder()
sage: unexp = ANUnaryExpr(sqrt(AA(2)), '~')
sage: unexp.handle_sage_input(sib, False, False)
({unop:~ {call: {atomic:sqrt}({call: {atomic:AA}({atomic:2})})}}, True)
sage: unexp.handle_sage_input(sib, False, True)
({call: {atomic:QQbar}({unop:~ {call: {atomic:sqrt}({call: {atomic:AA}({atomic:2})})}})}, Tr
```

is_complex()

Return whether or not this element is complex. Note that this is a data type check, and triggers no computations – if it returns False, the element might still be real, it just doesn't know it yet.

EXAMPLE:

```
sage: t = AA(sqrt(2))
sage: s = (-t)._descr
sage: s
<class 'sage.rings.qqbar.ANUnaryExpr'>
sage: s.is_complex()
False
sage: QQbar(-sqrt(2))._descr.is_complex()
True
```

kind()

Return a string describing what kind of element this is.

EXAMPLE:

```
sage: x = -QQbar(sqrt(2))
sage: y = x._descr
sage: type(y)
<class 'sage.rings.qqbar.ANUnaryExpr'>
sage: y.kind()
'other'
```

class sage.rings.qqbar.**AlgebraicField**

Bases: sage.misc.fast_methods.Singleton, sage.rings.qqbar.AlgebraicField_common

The field of all algebraic complex numbers.

algebraic_closure()

Return the algebraic closure of this field. As this field is already algebraically closed, just returns self.

EXAMPLES:

```
sage: QQbar.algebraic_closure()
Algebraic Field
```

completion(*p*, *prec*, *extras={}*)

Return the completion of self at the place p. Only implemented for $p = \infty$ at present.

INPUT:

- p – either a prime (not implemented at present) or Infinity
- prec – precision of approximate field to return
- extras – a dict of extra keyword arguments for the RealField constructor

EXAMPLES:

```
sage: QQbar.completion(infinity, 500)
Complex Field with 500 bits of precision
sage: QQbar.completion(infinity, prec=53, extras={'type':'RDF'})
Complex Double Field
sage: QQbar.completion(infinity, 53) is CC
True
sage: QQbar.completion(3, 20)
Traceback (most recent call last):
...
NotImplementedError
```

construction()

Return a functor that constructs self (used by the coercion machinery).

EXAMPLE:

```
sage: QQbar.construction()
(AlgebraicClosureFunctor, Rational Field)
```

gen(*n=0*)

Return the n-th element of the tuple returned by gens().

EXAMPLE:

```
sage: QQbar.gen(0)
1*I
sage: QQbar.gen(1)
Traceback (most recent call last):
...
IndexError: n must be 0
```

gens()

Return a set of generators for this field. As this field is not finitely generated over its prime field, we opt for just returning I.

EXAMPLE:

```
sage: QQbar.gens()
(1*I,)
```

ngens()

Return the size of the tuple returned by gens().

EXAMPLE:

```
sage: QQbar.ngens()
1
```

polynomial_root(*poly*, *interval*, *multiplicity=1*)

Given a polynomial with algebraic coefficients and an interval enclosing exactly one root of the polynomial, constructs an algebraic real representation of that root.

The polynomial need not be irreducible, or even squarefree; but if the given root is a multiple root, its multiplicity must be specified. (IMPORTANT NOTE: Currently, multiplicity-k roots are handled by taking

the $(k-1)$-st derivative of the polynomial. This means that the interval must enclose exactly one root of this derivative.)

The conditions on the arguments (that the interval encloses exactly one root, and that multiple roots match the given multiplicity) are not checked; if they are not satisfied, an error may be thrown (possibly later, when the algebraic number is used), or wrong answers may result.

Note that if you are constructing multiple roots of a single polynomial, it is better to use `QQbar.common_polynomial` to get a shared polynomial.

EXAMPLES:

```
sage: x = polygen(QQbar)
sage: phi = QQbar.polynomial_root(x^2 - x - 1, RIF(0, 2)); phi
1.6180339887498895?
sage: p = (x-1)^7 * (x-2)
sage: r = QQbar.polynomial_root(p, RIF(9/10, 11/10), multiplicity=7)
sage: r; r == 1
1
True
sage: p = (x-phi)*(x-sqrt(QQbar(2)))
sage: r = QQbar.polynomial_root(p, RIF(1, 3/2))
sage: r; r == sqrt(QQbar(2))
1.4142135623730951?
True
```

random_element (*poly_degree=2*, **args*, ***kwds*)

Returns a random algebraic number.

INPUT:

- `poly_degree` - default: 2 - degree of the random polynomial over the integers of which the returned algebraic number is a root. This is not necessarily the degree of the minimal polynomial of the number. Increase this parameter to achieve a greater diversity of algebraic numbers, at a cost of greater computation time. You can also vary the distribution of the coefficients but that will not vary the degree of the extension containing the element.
- `args`, `kwds` - arguments and keywords passed to the random number generator for elements of `ZZ`, the integers. See `random_element()` for details, or see example below.

OUTPUT:

An element of `QQbar`, the field of algebraic numbers (see `sage.rings.qqbar`).

ALGORITHM:

A polynomial with degree between 1 and `poly_degree`, with random integer coefficients is created. A root of this polynomial is chosen at random. The default degree is 2 and the integer coefficients come from a distribution heavily weighted towards $0, \pm 1, \pm 2$.

EXAMPLES:

```
sage: a = QQbar.random_element()
sage: a                              # random
0.2626138748742799? + 0.8769062830975992?*I
sage: a in QQbar
True

sage: b = QQbar.random_element(poly_degree=20)
sage: b                              # random
-0.8642649077479498? - 0.5995098147478391?*I
sage: b in QQbar
True
```

Parameters for the distribution of the integer coefficients of the polynomials can be passed on to the random element method for integers. For example, current default behavior of this method returns zero about 15% of the time; if we do not include zero as a possible coefficient, there will never be a zero constant term, and thus never a zero root.

```
sage: z = [QQbar.random_element(x=1, y=10) for _ in range(20)]
sage: QQbar(0) in z
False
```

```
If you just want real algebraic numbers you can filter them out.
Using an odd degree for the polynomials will insure some degree of
success. ::
```

```
sage: r = []
sage: while len(r) < 3:
...     x = QQbar.random_element(poly_degree=3)
...     if x in AA:
...       r.append(x)
sage: (len(r) == 3) and all([z in AA for z in r])
True
```

TESTS:

sage: QQbar.random_element('junk') Traceback (most recent call last): ... TypeError: polynomial degree must be an integer, not junk sage: QQbar.random_element(poly_degree=0) Traceback (most recent call last): ... ValueError: polynomial degree must be greater than zero, not 0

Random vectors already have a 'degree' keyword, so we cannot use that for the polynomial's degree.

```
sage: v = random_vector(QQbar, degree=2, poly_degree=3)
sage: v                                     # random
(0.4694381338921299?, -0.500000000000000? + 0.866025403784439?*I)
```

zeta(*n=4*)

Returns a primitive n'th root of unity, specifically $\exp(2 * \pi * i/n)$.

INPUT:

- n (integer) – default 4

EXAMPLES:

```
sage: QQbar.zeta(1)
1
sage: QQbar.zeta(2)
-1
sage: QQbar.zeta(3)
-0.500000000000000? + 0.866025403784439?*I
sage: QQbar.zeta(4)
1*I
sage: QQbar.zeta()
1*I
sage: QQbar.zeta(5)
0.3090169943749474? + 0.9510565162951536?*I
sage: QQbar.zeta(314159)
0.9999999997999997? + 0.00002000001689195824?*I
```

class sage.rings.qqbar.**AlgebraicField_common**

Bases: sage.rings.ring.Field

Common base class for the classes AlgebraicRealField and AlgebraicField.

characteristic()

Return the characteristic of this field. Since this class is only used for fields of characteristic 0, always returns 0.

EXAMPLES:

```
sage: AA.characteristic()
0
```

common_polynomial(*poly*)

Given a polynomial with algebraic coefficients, returns a wrapper that caches high-precision calculations and factorizations. This wrapper can be passed to polynomial_root in place of the polynomial.

Using common_polynomial makes no semantic difference, but will improve efficiency if you are dealing with multiple roots of a single polynomial.

EXAMPLES:

```
sage: x = polygen(ZZ)
sage: p = AA.common_polynomial(x^2 - x - 1)
sage: phi = AA.polynomial_root(p, RIF(1, 2))
sage: tau = AA.polynomial_root(p, RIF(-1, 0))
sage: phi + tau == 1
True
sage: phi * tau == -1
True

sage: x = polygen(SR)
sage: p = (x - sqrt(-5)) * (x - sqrt(3)); p
x^2 + (-sqrt(3) - sqrt(-5))*x + sqrt(3)*sqrt(-5)
sage: p = QQbar.common_polynomial(p)
sage: a = QQbar.polynomial_root(p, CIF(RIF(-0.1, 0.1), RIF(2, 3))); a
0.?e-18 + 2.236067977499790?*I
sage: b = QQbar.polynomial_root(p, RIF(1, 2)); b
1.732050807568878?
```

These "common polynomials" can be shared between real and complex roots:

```
sage: p = AA.common_polynomial(x^3 - x - 1)
sage: r1 = AA.polynomial_root(p, RIF(1.3, 1.4)); r1
1.324717957244746?
sage: r2 = QQbar.polynomial_root(p, CIF(RIF(-0.7, -0.6), RIF(0.5, 0.6))); r2
-0.6623589786223730? + 0.5622795120623013?*I
```

default_interval_prec()

Return the default interval precision used for root isolation.

EXAMPLES:

```
sage: AA.default_interval_prec()
64
```

is_finite()

Check whether this field is finite. Since this class is only used for fields of characteristic 0, always returns False.

EXAMPLE:

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

order()
Return the cardinality of self. Since this class is only used for fields of characteristic 0, always returns Infinity.

EXAMPLE:

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

class sage.rings.qqbar.**AlgebraicGenerator**(*field*, *root*)
Bases: sage.structure.sage_object.SageObject

An AlgebraicGenerator represents both an algebraic number α and the number field $\mathbf{Q}[\alpha]$. There is a single AlgebraicGenerator representing $\mathbf{Q}$ (with $\alpha = 0$).

The AlgebraicGenerator class is private, and should not be used directly.

conjugate()
If this generator is for the algebraic number α, return a generator for the complex conjugate of α.

EXAMPLE:

```
sage: from sage.rings.qqbar import AlgebraicGenerator
sage: x = polygen(QQ); f = x^4 + x + 17
sage: nf = NumberField(f,name='a')
sage: b = f.roots(QQbar)[0][0]
sage: root = b._descr
sage: gen = AlgebraicGenerator(nf, root)
sage: gen.conjugate()
Number Field in a with defining polynomial x^4 + x + 17 with a in -1.436449997483091? + 1.37
```

field()
Return the number field attached to self.

EXAMPLE:

```
sage: from sage.rings.qqbar import qq_generator, cyclotomic_generator
sage: qq_generator.field()
Rational Field
sage: cyclotomic_generator(3).field()
Cyclotomic Field of order 3 and degree 2
```

is_complex()
Return True if this is a generator for a non-real number field.

EXAMPLE:

```
sage: sage.rings.qqbar.cyclotomic_generator(7).is_complex()
True
sage: sage.rings.qqbar.qq_generator.is_complex()
False
```

is_trivial()
Returns true iff this is the trivial generator (alpha == 1), which does not actually extend the rationals.

EXAMPLES:

```
sage: from sage.rings.qqbar import qq_generator
sage: qq_generator.is_trivial()
True
```

pari_field()

Return the PARI field attached to this generator.

EXAMPLE:

```
sage: from sage.rings.qqbar import qq_generator
sage: qq_generator.pari_field()
Traceback (most recent call last):
...
ValueError: No PARI field attached to trivial generator

sage: from sage.rings.qqbar import ANRoot, AlgebraicGenerator, qq_generator
sage: y = polygen(QQ)
sage: x = polygen(QQbar)
sage: nf = NumberField(y^2 - y - 1, name='a', check=False)
sage: root = ANRoot(x^2 - x - 1, RIF(1, 2))
sage: gen = AlgebraicGenerator(nf, root)
sage: gen.pari_field()
 [y^2 - y - 1, [2, 0], ...]
```

root_as_algebraic()

Return the root attached to self as an algebraic number.

EXAMPLE:

```
sage: t = sage.rings.qqbar.qq_generator.root_as_algebraic(); t
1
sage: t.parent()
Algebraic Real Field
```

set_cyclotomic(*n*)

Store the fact that this is generator for a cyclotomic field.

EXAMPLE:

```
sage: y = sage.rings.qqbar.cyclotomic_generator(5) # indirect doctest
sage: y._cyclotomic
True
```

super_poly(*super*, *checked=None*)

Given a generator `gen` and another generator `super`, where `super` is the result of a tree of `union()` operations where one of the leaves is `gen`, `gen.super_poly(super)` returns a polynomial expressing the value of `gen` in terms of the value of `super` (except that if `gen` is `qq_generator`, `super_poly()` always returns None.)

EXAMPLES:

```
sage: from sage.rings.qqbar import AlgebraicGenerator, ANRoot, qq_generator
sage: _.<y> = QQ['y']
sage: x = polygen(QQbar)
sage: nf2 = NumberField(y^2 - 2, name='a', check=False)
sage: root2 = ANRoot(x^2 - 2, RIF(1, 2))
sage: gen2 = AlgebraicGenerator(nf2, root2)
sage: gen2
Number Field in a with defining polynomial y^2 - 2 with a in 1.4142135623730951?
sage: nf3 = NumberField(y^2 - 3, name='a', check=False)
```

```
sage: root3 = ANRoot(x^2 - 3, RIF(1, 2))
sage: gen3 = AlgebraicGenerator(nf3, root3)
sage: gen3
Number Field in a with defining polynomial y^2 - 3 with a in 1.732050807568878?
sage: gen2_3 = gen2.union(gen3)
sage: gen2_3
Number Field in a with defining polynomial y^4 - 4*y^2 + 1 with a in 0.5176380902050415?
sage: qq_generator.super_poly(gen2) is None
True
sage: gen2.super_poly(gen2_3)
-a^3 + 3*a
sage: gen3.super_poly(gen2_3)
-a^2 + 2
```

union(*other*)

Given generators `alpha` and `beta`, `alpha.union(beta)` gives a generator for the number field $\mathbf{Q}[\alpha][\beta]$.

EXAMPLES:

```
sage: from sage.rings.qqbar import ANRoot, AlgebraicGenerator, qq_generator
sage: _.<y> = QQ['y']
sage: x = polygen(QQbar)
sage: nf2 = NumberField(y^2 - 2, name='a', check=False)
sage: root2 = ANRoot(x^2 - 2, RIF(1, 2))
sage: gen2 = AlgebraicGenerator(nf2, root2)
sage: gen2
Number Field in a with defining polynomial y^2 - 2 with a in 1.414213562373095?
sage: nf3 = NumberField(y^2 - 3, name='a', check=False)
sage: root3 = ANRoot(x^2 - 3, RIF(1, 2))
sage: gen3 = AlgebraicGenerator(nf3, root3)
sage: gen3
Number Field in a with defining polynomial y^2 - 3 with a in 1.732050807568878?
sage: gen2.union(qq_generator) is gen2
True
sage: qq_generator.union(gen3) is gen3
True
sage: gen2.union(gen3)
Number Field in a with defining polynomial y^4 - 4*y^2 + 1 with a in 0.5176380902050415?
```

class sage.rings.qqbar.**AlgebraicGeneratorRelation**(*child1*, *child1_poly*, *child2*, *child2_poly*, *parent*)

Bases: sage.structure.sage_object.SageObject

A simple class for maintaining relations in the lattice of algebraic extensions.

class sage.rings.qqbar.**AlgebraicNumber**(*x*)

Bases: sage.rings.qqbar.AlgebraicNumber_base

The class for algebraic numbers (complex numbers which are the roots of a polynomial with integer coefficients). Much of its functionality is inherited from `AlgebraicNumber_base`.

complex_exact(*field*)

Given a `ComplexField`, return the best possible approximation of this number in that field. Note that if either component is sufficiently close to the halfway point between two floating-point numbers in the corresponding `RealField`, then this will trigger exact computation, which may be very slow.

EXAMPLES:

```
sage: a = QQbar.zeta(9) + I + QQbar.zeta(9).conjugate(); a
1.532088886237957? + 1.000000000000000?*I
sage: a.complex_exact(CIF)
1.532088886237957? + 1*I
```

complex_number (*field*)

Given a ComplexField, compute a good approximation to self in that field. The approximation will be off by at most two ulp's in each component, except for components which are very close to zero, which will have an absolute error at most 2**(-(field.prec()-1)).

EXAMPLES:

```
sage: a = QQbar.zeta(5)
sage: a.complex_number(CIF)
0.309016994374947 + 0.951056516295154*I
sage: (a + a.conjugate()).complex_number(CIF)
0.618033988749895 - 5.42101086242752e-20*I
```

conjugate()

Returns the complex conjugate of self.

EXAMPLES:

```
sage: QQbar(3 + 4*I).conjugate()
3 - 4*I
sage: QQbar.zeta(7).conjugate()
0.6234898018587335? - 0.7818314824680299?*I
sage: QQbar.zeta(7) + QQbar.zeta(7).conjugate()
1.246979603717467? + 0.?e-18*I
```

imag()

Return the imaginary part of self.

EXAMPLE:

```
sage: QQbar.zeta(7).imag()
0.7818314824680299?
```

interval_exact (*field*)

Given a ComplexIntervalField, compute the best possible approximation of this number in that field. Note that if either the real or imaginary parts of this number are sufficiently close to some floating-point number (and, in particular, if either is exactly representable in floating-point), then this will trigger exact computation, which may be very slow.

EXAMPLES:

```
sage: a = QQbar(I).sqrt(); a
0.7071067811865475? + 0.7071067811865475?*I
sage: a.interval_exact(CIF)
0.7071067811865475? + 0.7071067811865475?*I
sage: b = QQbar((1+I)*sqrt(2)/2)
sage: (a - b).interval(CIF)
0.?e-19 + 0.?e-18*I
sage: (a - b).interval_exact(CIF)
0
```

multiplicative_order()

Compute the multiplicative order of this algebraic real number. That is, find the smallest positive integer n such that $x^n = 1$. If there is no such n, returns +Infinity.

We first check that abs(x) is very close to 1. If so, we compute x exactly and examine its argument.

EXAMPLES:

```
sage: QQbar(-sqrt(3)/2 - I/2).multiplicative_order()
12
sage: QQbar(1).multiplicative_order()
1
sage: QQbar(-I).multiplicative_order()
4
sage: QQbar(707/1000 + 707/1000*I).multiplicative_order()
+Infinity
sage: QQbar(3/5 + 4/5*I).multiplicative_order()
+Infinity
```

norm()

Returns `self * self.conjugate()`. This is the algebraic definition of norm, if we view `QQbar` as `AA[I]`.

EXAMPLES:

```
sage: QQbar(3 + 4*I).norm()
25
sage: type(QQbar(I).norm())
<class 'sage.rings.qqbar.AlgebraicReal'>
sage: QQbar.zeta(1007).norm()
1
```

rational_argument()

Returns the argument of self, divided by 2π, as long as this result is rational. Otherwise returns None. Always triggers exact computation.

EXAMPLES:

```
sage: QQbar((1+I)*(sqrt(2)+sqrt(5))).rational_argument()
1/8
sage: QQbar(-1 + I*sqrt(3)).rational_argument()
1/3
sage: QQbar(-1 - I*sqrt(3)).rational_argument()
-1/3
sage: QQbar(3+4*I).rational_argument() is None
True
sage: (QQbar.zeta(7654321)^65536).rational_argument()
65536/7654321
sage: (QQbar.zeta(3)^65536).rational_argument()
1/3
```

real()

Return the real part of self.

EXAMPLE:

```
sage: QQbar.zeta(5).real()
0.3090169943749474?
```

class sage.rings.qqbar.**AlgebraicNumber_base**(*parent*, *x*)

Bases: sage.structure.element.FieldElement

This is the common base class for algebraic numbers (complex numbers which are the zero of a polynomial in $\mathbf{Z}[x]$) and algebraic reals (algebraic numbers which happen to be real).

`AlgebraicNumber` objects can be created using `QQbar` (== `AlgebraicNumberField()`), and `AlgebraicReal` objects can be created using `AA` (== `AlgebraicRealField()`). They can be created

either by coercing a rational or a symbolic expression, or by using the `QQbar.polynomial_root()` or `AA.polynomial_root()` method to construct a particular root of a polynomial with algebraic coefficients. Also, `AlgebraicNumber` and `AlgebraicReal` are closed under addition, subtraction, multiplication, division (except by 0), and rational powers (including roots), except that for a negative `AlgebraicReal`, taking a power with an even denominator returns an `AlgebraicNumber` instead of an `AlgebraicReal`.

`AlgebraicNumber` and `AlgebraicReal` objects can be approximated to any desired precision. They can be compared exactly; if the two numbers are very close, or are equal, this may require exact computation, which can be extremely slow.

As long as exact computation is not triggered, computation with algebraic numbers should not be too much slower than computation with intervals. As mentioned above, exact computation is triggered when comparing two algebraic numbers which are very close together. This can be an explicit comparison in user code, but the following list of actions (not necessarily complete) can also trigger exact computation:

- Dividing by an algebraic number which is very close to 0.
- Using an algebraic number which is very close to 0 as the leading coefficient in a polynomial.
- Taking a root of an algebraic number which is very close to 0.

The exact definition of "very close" is subject to change; currently, we compute our best approximation of the two numbers using 128-bit arithmetic, and see if that's sufficient to decide the comparison. Note that comparing two algebraic numbers which are actually equal will always trigger exact computation, unless they are actually the same object.

EXAMPLES:

```
sage: sqrt(QQbar(2))
1.414213562373095?
sage: sqrt(QQbar(2))^2 == 2
True
sage: x = polygen(QQbar)
sage: phi = QQbar.polynomial_root(x^2 - x - 1, RIF(1, 2))
sage: phi
1.618033988749895?
sage: phi^2 == phi+1
True
sage: AA(sqrt(65537))
256.0019531175495?
```

as_number_field_element(*minimal=False*)

Returns a number field containing this value, a representation of this value as an element of that number field, and a homomorphism from the number field back to `AA` or `QQbar`.

This may not return the smallest such number field, unless `minimal=True` is specified.

To compute a single number field containing multiple algebraic numbers, use the function `number_field_elements_from_algebraics` instead.

EXAMPLES:

```
sage: QQbar(sqrt(8)).as_number_field_element()
(Number Field in a with defining polynomial y^2 - 2, 2*a, Ring morphism:
    From: Number Field in a with defining polynomial y^2 - 2
    To:   Algebraic Real Field
    Defn: a |--> 1.414213562373095?)
sage: x = polygen(ZZ)
sage: p = x^3 + x^2 + x + 17
sage: (rt,) = p.roots(ring=AA, multiplicities=False); rt
-2.804642726932742?
sage: (nf, elt, hom) = rt.as_number_field_element()
```

```
sage: nf, elt, hom
(Number Field in a with defining polynomial y^3 - 2*y^2 - 31*y - 50, a^2 - 5*a - 19, Ring mo
  From: Number Field in a with defining polynomial y^3 - 2*y^2 - 31*y - 50
  To:   Algebraic Real Field
  Defn: a |--> 7.2376531398011042?)
sage: hom(elt) == rt
True
```

We see an example where we do not get the minimal number field unless we specify `minimal=True`:

```
sage: rt2 = AA(sqrt(2))
sage: rt3 = AA(sqrt(3))
sage: rt3b = rt2 + rt3 - rt2
sage: rt3b.as_number_field_element()
(Number Field in a with defining polynomial y^4 - 4*y^2 + 1, -a^2 + 2, Ring morphism:
    From: Number Field in a with defining polynomial y^4 - 4*y^2 + 1
    To:   Algebraic Real Field
    Defn: a |--> 0.51763809020504152?)
sage: rt3b.as_number_field_element(minimal=True)
(Number Field in a with defining polynomial y^2 - 3, a, Ring morphism:
    From: Number Field in a with defining polynomial y^2 - 3
    To:   Algebraic Real Field
    Defn: a |--> 1.7320508075688782?)
```

degree()

Return the degree of this algebraic number (the degree of its minimal polynomial, or equivalently, the degree of the smallest algebraic extension of the rationals containing this number).

EXAMPLES:

```
sage: QQbar(5/3).degree()
1
sage: sqrt(QQbar(2)).degree()
2
sage: QQbar(17).nth_root(5).degree()
5
sage: sqrt(3+sqrt(QQbar(8))).degree()
2
```

exactify()

Compute an exact representation for this number.

EXAMPLES:

```
sage: two = QQbar(4).nth_root(4)^2
sage: two
2.0000000000000000?
sage: two.exactify()
sage: two
2
```

interval(*field*)

Given an interval field (real or complex, as appropriate) of precision p, compute an interval representation of self with `diameter()` at most 2^{-p}; then round that representation into the given field. Here `diameter()` is relative diameter for intervals not containing 0, and absolute diameter for intervals that do contain 0; thus, if the returned interval does not contain 0, it has at least $p - 1$ good bits.

EXAMPLES:

```
sage: RIF64 = RealIntervalField(64)
sage: x = AA(2).sqrt()
sage: y = x*x
sage: y = 1000 * y - 999 * y
sage: y.interval_fast(RIF64)
2.000000000000000?
sage: y.interval(RIF64)
2.0000000000000000?
sage: CIF64 = ComplexIntervalField(64)
sage: x = QQbar.zeta(11)
sage: x.interval_fast(CIF64)
0.84125353283118116897 + 0.54064081745559758?*I
sage: x.interval(CIF64)
0.84125353283118116897 + 0.54064081745559758?*I
```

interval_diameter(*diam*)

Compute an interval representation of self with `diameter()` at most `diam`. The precision of the returned value is unpredictable.

EXAMPLES:

```
sage: AA(2).sqrt().interval_diameter(1e-10)
1.41421356237309504?
sage: AA(2).sqrt().interval_diameter(1e-30)
1.41421356237309504880168872420969807857?
sage: QQbar(2).sqrt().interval_diameter(1e-10)
1.41421356237309504?
sage: QQbar(2).sqrt().interval_diameter(1e-30)
1.41421356237309504880168872420969807857?
```

interval_fast(*field*)

Given a `RealIntervalField`, compute the value of this number using interval arithmetic of at least the precision of the field, and return the value in that field. (More precision may be used in the computation.) The returned interval may be arbitrarily imprecise, if this number is the result of a sufficiently long computation chain.

EXAMPLES:

```
sage: x = AA(2).sqrt()
sage: x.interval_fast(RIF)
1.414213562373095?
sage: x.interval_fast(RealIntervalField(200))
1.41421356237309504880168872420969807856967187537694807317667?
sage: x = QQbar(I).sqrt()
sage: x.interval_fast(CIF)
0.7071067811865475? + 0.7071067811865475?*I
sage: x.interval_fast(RIF)
Traceback (most recent call last):
...
TypeError: Unable to convert number to real interval.
```

is_integer()

Return True if this number is a integer

EXAMPLES:

```
sage: QQbar(2).is_integer()
True
sage: QQbar(1/2).is_integer()
False
```

is_square()
Return whether or not this number is square.

OUTPUT:

(boolean) True in all cases for elements of QQbar; True for non-negative elements of AA, otherwise False.

EXAMPLES:

```
sage: AA(2).is_square()
True
sage: AA(-2).is_square()
False
sage: QQbar(-2).is_square()
True
sage: QQbar(I).is_square()
True
```

minpoly()
Compute the minimal polynomial of this algebraic number. The minimal polynomial is the monic polynomial of least degree having this number as a root; it is unique.

EXAMPLES:

```
sage: QQbar(4).sqrt().minpoly()
x - 2
sage: ((QQbar(2).nth_root(4))^2).minpoly()
x^2 - 2
sage: v = sqrt(QQbar(2)) + sqrt(QQbar(3)); v
3.146264369941973?
sage: p = v.minpoly(); p
x^4 - 10*x^2 + 1
sage: p(RR(v.real()))
1.31006316905768e-14
```

nth_root(n)
Return the n-th root of this number.

Note that for odd n and negative real numbers, `AlgebraicReal` and `AlgebraicNumber` values give different answers: `AlgebraicReal` values prefer real results, and `AlgebraicNumber` values return the principal root.

EXAMPLES:

```
sage: AA(-8).nth_root(3)
-2
sage: QQbar(-8).nth_root(3)
1.000000000000000? + 1.732050807568878?*I
sage: QQbar.zeta(12).nth_root(15)
0.9993908270190957? + 0.03489949670250097?*I
```

radical_expression()
Attempt to obtain a symbolic expression using radicals. If no exact symbolic expression can be found, the algebraic number will be returned without modification.

EXAMPLES:

```
sage: AA(1/sqrt(5)).radical_expression()
1/5*sqrt(5)
sage: AA(sqrt(5 + sqrt(5))).radical_expression()
sqrt(sqrt(5) + 5)
```

```
sage: QQbar.zeta(5).radical_expression()
1/4*sqrt(5) + 1/2*sqrt(-1/2*sqrt(5) - 5/2) - 1/4
sage: a = QQ[x](x^7 - x - 1).roots(AA, False)[0]
sage: a.radical_expression()
1.112775684278706?
sage: a.radical_expression().parent() == SR
False
sage: a = sorted(QQ[x](x^7-x-1).roots(QQbar, False), key=imag)[0]
sage: a.radical_expression()
-0.3636235193291805? - 0.9525611952610331?*I
sage: QQbar.zeta(5).imag().radical_expression()
1/2*sqrt(1/2*sqrt(5) + 5/2)
sage: AA(5/3).radical_expression()
5/3
sage: AA(5/3).radical_expression().parent() == SR
True
sage: QQbar(0).radical_expression()
0
```

TESTS:

In this example we find the correct answer despite the fact that multiple roots overlap with the current value. As a consequence, the precision of the evaluation will have to be increased.

```
sage: a = AA(sqrt(2) + 10^25)
sage: p = a.minpoly()
sage: v = a._value
sage: f = ComplexIntervalField(v.prec())
sage: [f(b.rhs()).overlaps(f(v)) for b in SR(p).solve(x)]
[True, True]
sage: a.radical_expression()
sqrt(2) + 10000000000000000000000000
```

simplify()

Compute an exact representation for this number, in the smallest possible number field.

EXAMPLES:

```
sage: rt2 = AA(sqrt(2))
sage: rt3 = AA(sqrt(3))
sage: rt2b = rt3 + rt2 - rt3
sage: rt2b.exactify()
sage: rt2b._exact_value()
a^3 - 3*a where a^4 - 4*a^2 + 1 = 0 and a in 1.9318516525781377?
sage: rt2b.simplify()
sage: rt2b._exact_value()
a where a^2 - 2 = 0 and a in 1.4142135623730951?
```

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

Return the square root(s) of this number.

INPUT:

- `extend` - bool (default: True); ignored if self is in QQbar, or positive in AA. If self is negative in AA, do the following: if True, return a square root of self in QQbar, otherwise raise a ValueError.
- `all` - bool (default: False); if True, return a list of all square roots. If False, return just one square root, or raise an ValueError if self is a negative element of AA and extend=False.

OUTPUT:

Either the principal square root of self, or a list of its square roots (with the principal one first).

EXAMPLES:

```
sage: AA(2).sqrt()
1.4142135623730951?

sage: QQbar(I).sqrt()
0.7071067811865475? + 0.7071067811865475?*I
sage: QQbar(I).sqrt(all=True)
[0.7071067811865475? + 0.7071067811865475?*I, -0.7071067811865475? - 0.7071067811865475?*I]

sage: a = QQbar(0)
sage: a.sqrt()
0
sage: a.sqrt(all=True)
[0]

sage: a = AA(0)
sage: a.sqrt()
0
sage: a.sqrt(all=True)
[0]
```

This second example just shows that the program doesn't care where 0 is defined, it gives the same answer regardless. After all, how many ways can you square-root zero?

```
sage: AA(-2).sqrt()
1.4142135623730951?*I

sage: AA(-2).sqrt(all=True)
[1.4142135623730951?*I, -1.4142135623730951?*I]

sage: AA(-2).sqrt(extend=False)
Traceback (most recent call last):
...
ValueError: -2 is not a square in AA, being negative. Use extend = True for a square root in
```

class sage.rings.qqbar.**AlgebraicPolynomialTracker**(*poly*)

Bases: sage.structure.sage_object.SageObject

Keeps track of a polynomial used for algebraic numbers.

If multiple algebraic numbers are created as roots of a single polynomial, this allows the polynomial and information about the polynomial to be shared. This reduces work if the polynomial must be recomputed at higher precision, or if it must be factored.

This class is private, and should only be constructed by `AA.common_polynomial()` or `QQbar.common_polynomial()`, and should only be used as an argument to `AA.polynomial_root()` or `QQbar.polynomial_root()`. (It doesn't matter whether you create the common polynomial with `AA.common_polynomial()` or `QQbar.common_polynomial()`.)

EXAMPLES:

```
sage: x = polygen(QQbar)
sage: P = QQbar.common_polynomial(x^2 - x - 1)
sage: P
x^2 - x - 1
sage: QQbar.polynomial_root(P, RIF(1, 2))
1.618033988749895?
```

complex_roots(*prec*, *multiplicity*)

Find the roots of self in the complex field to precision prec.

EXAMPLES:

```
sage: x = polygen(ZZ)
sage: cp = AA.common_polynomial(x^4 - 2)
```

Note that the precision is not guaranteed to find the tightest possible interval since complex_roots() depends on the underlying BLAS implementation.

```
sage: cp.complex_roots(30, 1)
[-1.18920711500272...?,
 1.189207115002721?,
 -1.189207115002721?*I,
 1.189207115002721?*I]
```

exactify()

Compute a common field that holds all of the algebraic coefficients of this polynomial, then factor the polynomial over that field. Store the factors for later use (ignoring multiplicity).

EXAMPLE:

```
sage: x = polygen(AA)
sage: p = sqrt(AA(2)) * x^2 - sqrt(AA(3))
sage: cp = AA.common_polynomial(p)
sage: cp._exact
False
sage: cp.exactify()
sage: cp._exact
True
```

factors()

EXAMPLE:

```
sage: x=polygen(QQ); f=QQbar.common_polynomial(x^4 + 4)
sage: f.factors()
[y^2 - 2*y + 2, y^2 + 2*y + 2]
```

generator()

Return an AlgebraicGenerator for a number field containing all the coefficients of self.

EXAMPLE:

```
sage: x = polygen(AA)
sage: p = sqrt(AA(2)) * x^2 - sqrt(AA(3))
sage: cp = AA.common_polynomial(p)
sage: cp.generator()
Number Field in a with defining polynomial y^4 - 4*y^2 + 1 with a in 1.931851652578137?
```

is_complex()

Return True if the coefficients of this polynomial are non-real.

EXAMPLE:

```
sage: x = polygen(QQ); f = x^3 - 7
sage: g = AA.common_polynomial(f)
sage: g.is_complex()
False
sage: QQbar.common_polynomial(x^3 - QQbar(I)).is_complex()
True
```

poly()
Return the underlying polynomial of self.

EXAMPLE:

```
sage: x = polygen(QQ); f = x^3 - 7
sage: g = AA.common_polynomial(f)
sage: g.poly() == f
True
```

class sage.rings.qqbar.**AlgebraicReal**(*x*)
Bases: sage.rings.qqbar.AlgebraicNumber_base

Create an algebraic real from x, possibly taking the real part of x.

TESTS:

Both of the following examples, from trac ticket #11728, trigger taking the real part below. This is necessary because sometimes a very small (e.g., 1e-17) complex part appears in a complex interval used to create an AlgebraicReal.:

```
sage: a = QQbar((-1)^(1/4)); b = AA(a^3-a); t = b.as_number_field_element()
sage: b*1
-1.414213562373095?
```

ceil()
Return the smallest integer not smaller than `self`.

EXAMPLES:

```
sage: AA(sqrt(2)).ceil()
2
sage: AA(-sqrt(2)).ceil()
-1
sage: AA(42).ceil()
42
```

conjugate()
Returns the complex conjugate of self, i.e. returns itself.

EXAMPLES:

```
sage: a = AA(sqrt(2) + sqrt(3))
sage: a.conjugate()
3.146264369941973?
sage: a.conjugate() is a
True
```

floor()
Return the largest integer not greater than `self`.

EXAMPLES:

```
sage: AA(sqrt(2)).floor()
1
sage: AA(-sqrt(2)).floor()
-2
sage: AA(42).floor()
42
```

TESTS:

Check that trac ticket #15501 is fixed:

```
sage: a = QQbar((-1)^(1/4)).real()
sage: (floor(a-a) + a).parent()
Algebraic Real Field
```

imag()

Returns the imaginary part of this algebraic real (so it always returns 0).

EXAMPLES:

```
sage: a = AA(sqrt(2) + sqrt(3))
sage: a.imag()
0
sage: parent(a.imag())
Algebraic Real Field
```

interval_exact(*field*)

Given a `RealIntervalField`, compute the best possible approximation of this number in that field. Note that if this number is sufficiently close to some floating-point number (and, in particular, if this number is exactly representable in floating-point), then this will trigger exact computation, which may be very slow.

EXAMPLES:

```
sage: x = AA(2).sqrt()
sage: y = x*x
sage: x.interval(RIF)
1.414213562373095?
sage: x.interval_exact(RIF)
1.414213562373095?
sage: y.interval(RIF)
2.000000000000000?
sage: y.interval_exact(RIF)
2
sage: z = 1 + AA(2).sqrt() / 2^200
sage: z.interval(RIF)
1.000000000000001?
sage: z.interval_exact(RIF)
1.000000000000001?
```

real()

Returns the real part of this algebraic real (so it always returns self).

EXAMPLES:

```
sage: a = AA(sqrt(2) + sqrt(3))
sage: a.real()
3.146264369941973?
sage: a.real() is a
True
```

real_exact(*field*)

Given a `RealField`, compute the best possible approximation of this number in that field. Note that if this number is sufficiently close to the halfway point between two floating-point numbers in the field (for the default round-to-nearest mode) or if the number is sufficiently close to a floating-point number in the field (for directed rounding modes), then this will trigger exact computation, which may be very slow.

The rounding mode of the field is respected.

EXAMPLES:

```
sage: x = AA(2).sqrt()^2
sage: x.real_exact(RR)
2.00000000000000
sage: x.real_exact(RealField(53, rnd='RNDD'))
2.00000000000000
sage: x.real_exact(RealField(53, rnd='RNDU'))
2.00000000000000
sage: x.real_exact(RealField(53, rnd='RNDZ'))
2.00000000000000
sage: (-x).real_exact(RR)
-2.00000000000000
sage: (-x).real_exact(RealField(53, rnd='RNDD'))
-2.00000000000000
sage: (-x).real_exact(RealField(53, rnd='RNDU'))
-2.00000000000000
sage: (-x).real_exact(RealField(53, rnd='RNDZ'))
-2.00000000000000
sage: y = (x-2).real_exact(RR).abs()
sage: y == 0.0 or y == -0.0 # the sign of 0.0 is not significant in MPFI
True
sage: y = (x-2).real_exact(RealField(53, rnd='RNDD'))
sage: y == 0.0 or y == -0.0 # same as above
True
sage: y = (x-2).real_exact(RealField(53, rnd='RNDU'))
sage: y == 0.0 or y == -0.0 # idem
True
sage: y = (x-2).real_exact(RealField(53, rnd='RNDZ'))
sage: y == 0.0 or y == -0.0 # ibidem
True
sage: y = AA(2).sqrt()
sage: y.real_exact(RR)
1.41421356237310
sage: y.real_exact(RealField(53, rnd='RNDD'))
1.41421356237309
sage: y.real_exact(RealField(53, rnd='RNDU'))
1.41421356237310
sage: y.real_exact(RealField(53, rnd='RNDZ'))
1.41421356237309
```

real_number(*field*)

Given a `RealField`, compute a good approximation to self in that field. The approximation will be off by at most two ulp's, except for numbers which are very close to 0, which will have an absolute error at most `2**(-(field.prec()-1))`. Also, the rounding mode of the field is respected.

EXAMPLES:

```
sage: x = AA(2).sqrt()^2
sage: x.real_number(RR)
2.00000000000000
sage: x.real_number(RealField(53, rnd='RNDD'))
1.99999999999999
sage: x.real_number(RealField(53, rnd='RNDU'))
2.00000000000001
sage: x.real_number(RealField(53, rnd='RNDZ'))
1.99999999999999
sage: (-x).real_number(RR)
-2.00000000000000
sage: (-x).real_number(RealField(53, rnd='RNDD'))
-2.00000000000001
```

```
sage: (-x).real_number(RealField(53, rnd='RNDU'))
-1.99999999999999
sage: (-x).real_number(RealField(53, rnd='RNDZ'))
-1.99999999999999
sage: (x-2).real_number(RR)
5.42101086242752e-20
sage: (x-2).real_number(RealField(53, rnd='RNDD'))
-1.08420217248551e-19
sage: (x-2).real_number(RealField(53, rnd='RNDU'))
2.16840434497101e-19
sage: (x-2).real_number(RealField(53, rnd='RNDZ'))
0.000000000000000
sage: y = AA(2).sqrt()
sage: y.real_number(RR)
1.41421356237309
sage: y.real_number(RealField(53, rnd='RNDD'))
1.41421356237309
sage: y.real_number(RealField(53, rnd='RNDU'))
1.41421356237310
sage: y.real_number(RealField(53, rnd='RNDZ'))
1.41421356237309
```

round()

Round `self` to the nearest integer.

EXAMPLES:

```
sage: AA(sqrt(2)).round()
1
sage: AA(1/2).round()
1
sage: AA(-1/2).round()
-1
```

sign()

Compute the sign of this algebraic number (return -1 if negative, 0 if zero, or 1 if positive).

Computes an interval enclosing this number using 128-bit interval arithmetic; if this interval includes 0, then fall back to exact computation (which can be very slow).

EXAMPLES:

```
sage: AA(-5).nth_root(7).sign()
-1
sage: (AA(2).sqrt() - AA(2).sqrt()).sign()
0
```

trunc()

Round `self` to the nearest integer toward zero.

EXAMPLES:

```
sage: AA(sqrt(2)).trunc()
1
sage: AA(-sqrt(2)).trunc()
-1
sage: AA(1).trunc()
1
sage: AA(-1).trunc()
-1
```

class sage.rings.qqbar.**AlgebraicRealField**

Bases: sage.misc.fast_methods.Singleton, sage.rings.qqbar.AlgebraicField_common

The field of algebraic reals.

TESTS:

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

algebraic_closure()

Return the algebraic closure of this field, which is the field $\overline{\mathbf{Q}}$ of algebraic numbers.

EXAMPLES:

```
sage: AA.algebraic_closure()
Algebraic Field
```

completion(*p*, *prec*, *extras={}*)

Return the completion of self at the place p. Only implemented for $p = \infty$ at present.

INPUT:

- p – either a prime (not implemented at present) or Infinity
- prec – precision of approximate field to return
- extras – a dict of extra keyword arguments for the RealField constructor

EXAMPLES:

```
sage: AA.completion(infinity, 500)
Real Field with 500 bits of precision
sage: AA.completion(infinity, prec=53, extras={'type':'RDF'})
Real Double Field
sage: AA.completion(infinity, 53) is RR
True
sage: AA.completion(7, 10)
Traceback (most recent call last):
...
NotImplementedError
```

gen(*n=0*)

Return the n-th element of the tuple returned by gens().

EXAMPLE:

```
sage: AA.gen(0)
1
sage: AA.gen(1)
Traceback (most recent call last):
...
IndexError: n must be 0
```

gens()

Return a set of generators for this field. As this field is not finitely generated, we opt for just returning 1.

EXAMPLE:

```
sage: AA.gens()
(1,)
```

ngens()

Return the size of the tuple returned by gens().

EXAMPLE:

```
sage: AA.ngens()
1
```

polynomial_root (*poly*, *interval*, *multiplicity=1*)

Given a polynomial with algebraic coefficients and an interval enclosing exactly one root of the polynomial, constructs an algebraic real representation of that root.

The polynomial need not be irreducible, or even squarefree; but if the given root is a multiple root, its multiplicity must be specified. (IMPORTANT NOTE: Currently, multiplicity-k roots are handled by taking the $(k-1)$-st derivative of the polynomial. This means that the interval must enclose exactly one root of this derivative.)

The conditions on the arguments (that the interval encloses exactly one root, and that multiple roots match the given multiplicity) are not checked; if they are not satisfied, an error may be thrown (possibly later, when the algebraic number is used), or wrong answers may result.

Note that if you are constructing multiple roots of a single polynomial, it is better to use `AA.common_polynomial` (or `QQbar.common_polynomial`; the two are equivalent) to get a shared polynomial.

EXAMPLES:

```
sage: x = polygen(AA)
sage: phi = AA.polynomial_root(x^2 - x - 1, RIF(1, 2)); phi
1.618033988749895?
sage: p = (x-1)^7 * (x-2)
sage: r = AA.polynomial_root(p, RIF(9/10, 11/10), multiplicity=7)
sage: r; r == 1
1.000000000000000?
True
sage: p = (x-phi)*(x-sqrt(AA(2)))
sage: r = AA.polynomial_root(p, RIF(1, 3/2))
sage: r; r == sqrt(AA(2))
1.414213562373095?
True
```

We allow complex polynomials, as long as the particular root in question is real.

```
sage: K.<im> = QQ[I]
sage: x = polygen(K)
sage: p = (im + 1) * (x^3 - 2); p
(I + 1)*x^3 - 2*I - 2
sage: r = AA.polynomial_root(p, RIF(1, 2)); r^3
2.000000000000000?
```

zeta (*n=2*)

Return an n-th root of unity in this field. This will raise a `ValueError` if $n \neq \{1, 2\}$ since no such root exists.

INPUT:

- `n` (integer) – default 2

EXAMPLE:

```
sage: AA.zeta(1)
1
sage: AA.zeta(2)
-1
sage: AA.zeta()
```

```
-1
sage: AA.zeta(3)
Traceback (most recent call last):
...
ValueError: no n-th root of unity in algebraic reals
```

Some silly inputs:

```
sage: AA.zeta(Mod(-5, 7))
-1
sage: AA.zeta(0)
Traceback (most recent call last):
...
ValueError: no n-th root of unity in algebraic reals
```

sage.rings.qqbar.**an_addsub_element**(*a*, *b*, *sub*)

Add or subtract two elements represented as elements of number fields.

EXAMPLES:

```
sage: a = QQbar(sqrt(2) + sqrt(3)); a.exactify()
sage: b = QQbar(sqrt(3) + sqrt(5)); b.exactify()
sage: from sage.rings.qqbar import an_addsub_element
sage: an_addsub_element(a,b,False)
<class 'sage.rings.qqbar.ANBinaryExpr'>
```

sage.rings.qqbar.**an_addsub_expr**(*a*, *b*, *sub*)

Add or subtract algebraic numbers represented as multi-part expressions.

EXAMPLE:

```
sage: a = QQbar(sqrt(2)) + QQbar(sqrt(3))
sage: b = QQbar(sqrt(3)) + QQbar(sqrt(5))
sage: type(a._descr); type(b._descr)
<class 'sage.rings.qqbar.ANBinaryExpr'>
<class 'sage.rings.qqbar.ANBinaryExpr'>
sage: from sage.rings.qqbar import an_addsub_expr
sage: x = an_addsub_expr(a, b, False); x
<class 'sage.rings.qqbar.ANBinaryExpr'>
sage: x.exactify()
-6/7*a^7 + 2/7*a^6 + 71/7*a^5 - 26/7*a^4 - 125/7*a^3 + 72/7*a^2 + 43/7*a - 47/7 where a^8 - 12*a
```

sage.rings.qqbar.**an_addsub_gaussian**(*a*, *b*, *sub*)

Used to add and subtract algebraic numbers when both are in $\mathbf{Q}(i)$.

EXAMPLE:

```
sage: i = QQbar(I)
sage: from sage.rings.qqbar import an_addsub_gaussian
sage: x=an_addsub_gaussian(2 + 3*i, 2/3 + 1/4*i, True); x
11/4*I + 4/3 where a^2 + 1 = 0 and a in 1*I
sage: type(x)
<class 'sage.rings.qqbar.ANExtensionElement'>
```

sage.rings.qqbar.**an_addsub_rational**(*a*, *b*, *sub*)

Used to add and subtract algebraic numbers. Used when both are actually rational.

EXAMPLE:

```
sage: from sage.rings.qqbar import an_addsub_rational
sage: f = an_addsub_rational(QQbar(2), QQbar(3/7), False); f
```

```
17/7
sage: type(f)
<class 'sage.rings.qqbar.ANRational'>
```

sage.rings.qqbar.**an_addsub_rootunity**(*a*, *b*, *sub*)

Add or subtract two algebraic numbers represented as a rational multiple of a root of unity.

EXAMPLE:

```
sage: a = 2*QQbar.zeta(7)
sage: b = 3*QQbar.zeta(8)
sage: type(a._descr)
<class 'sage.rings.qqbar.ANRootOfUnity'>
sage: from sage.rings.qqbar import an_addsub_rootunity
sage: an_addsub_rootunity(a, b, False)
<class 'sage.rings.qqbar.ANBinaryExpr'>
sage: an_addsub_rootunity(a, 3*QQbar.zeta(7), True)
-1*e^(2*pi*I*1/7)
```

sage.rings.qqbar.**an_addsub_zero**(*a*, *b*, *sub*)

Used to add and subtract algebraic numbers. Used when one of a and b is zero.

EXAMPLES:

```
sage: from sage.rings.qqbar import an_addsub_zero
sage: f = an_addsub_zero(QQbar(sqrt(2)), QQbar(0), False); f
Root 1.4142135623730950488? of x^2 - 2
sage: type(f)
<class 'sage.rings.qqbar.ANRoot'>
sage: an_addsub_zero(QQbar(0), QQbar(sqrt(2)), True)
<class 'sage.rings.qqbar.ANUnaryExpr'>
```

sage.rings.qqbar.**an_muldiv_element**(*a*, *b*, *div*)

Multiply or divide two elements represented as elements of number fields.

EXAMPLES:

```
sage: a = QQbar(sqrt(2) + sqrt(3)); a.exactify()
sage: b = QQbar(sqrt(3) + sqrt(5)); b.exactify()
sage: type(a._descr)
<class 'sage.rings.qqbar.ANExtensionElement'>
sage: from sage.rings.qqbar import an_muldiv_element
sage: an_muldiv_element(a,b,False)
<class 'sage.rings.qqbar.ANBinaryExpr'>
```

sage.rings.qqbar.**an_muldiv_expr**(*a*, *b*, *div*)

Multiply or divide algebraic numbers represented as multi-part expressions.

EXAMPLE:

```
sage: a = QQbar(sqrt(2)) + QQbar(sqrt(3))
sage: b = QQbar(sqrt(3)) + QQbar(sqrt(5))
sage: type(a._descr)
<class 'sage.rings.qqbar.ANBinaryExpr'>
sage: from sage.rings.qqbar import an_muldiv_expr
sage: x = an_muldiv_expr(a, b, False); x
<class 'sage.rings.qqbar.ANBinaryExpr'>
sage: x.exactify()
2*a^7 - a^6 - 24*a^5 + 12*a^4 + 46*a^3 - 22*a^2 - 22*a + 9 where a^8 - 12*a^6 + 23*a^4 - 12*a^2
```

sage.rings.qqbar.**an_muldiv_gaussian**(*a*, *b*, *div*)
Used to multiply and divide algebraic numbers when both are in $\mathbf{Q}(i)$.

EXAMPLE:

```
sage: i = QQbar(I)
sage: from sage.rings.qqbar import an_muldiv_gaussian
sage: x=an_muldiv_gaussian(2 + 3*i, 2/3 + 1/4*i, True); x
216/73*I + 300/73 where a^2 + 1 = 0 and a in 1*I
sage: type(x)
<class 'sage.rings.qqbar.ANExtensionElement'>
```

sage.rings.qqbar.**an_muldiv_rational**(*a*, *b*, *div*)
Used to multiply and divide algebraic numbers. Used when both are actually rational.

EXAMPLE:

```
sage: from sage.rings.qqbar import an_muldiv_rational
sage: f = an_muldiv_rational(QQbar(2), QQbar(3/7), False); f
6/7
sage: type(f)
<class 'sage.rings.qqbar.ANRational'>
```

sage.rings.qqbar.**an_muldiv_rootunity**(*a*, *b*, *div*)
Multiply or divide two algebraic numbers represented as a rational multiple of a root of unity.

EXAMPLE:

```
sage: a = 2*QQbar.zeta(7)
sage: b = 3*QQbar.zeta(8)
sage: type(a._descr)
<class 'sage.rings.qqbar.ANRootOfUnity'>
sage: from sage.rings.qqbar import an_muldiv_rootunity
sage: an_muldiv_rootunity(a, b, True)
2/3*e^(2*pi*I*1/56)
```

sage.rings.qqbar.**an_muldiv_zero**(*a*, *b*, *div*)
Used to multiply and divide algebraic numbers. Used when one of a and b is zero.

EXAMPLES:

```
sage: from sage.rings.qqbar import an_muldiv_zero
sage: f = an_muldiv_zero(QQbar(sqrt(2)), QQbar(0), False); f
0
sage: type(f)
<class 'sage.rings.qqbar.ANRational'>
sage: an_muldiv_zero(QQbar(sqrt(2)), QQbar(sqrt(0)), True)
Traceback (most recent call last):
...
ValueError: algebraic number division by zero
```

sage.rings.qqbar.**clear_denominators**(*poly*)
Takes a monic polynomial and rescales the variable to get a monic polynomial with "integral" coefficients. Works on any univariate polynomial whose base ring has a `denominator()` method that returns integers; for example, the base ring might be $\mathbf{Q}$ or a number field.

Returns the scale factor and the new polynomial.

(Inspired by Pari's `primitive_pol_to_monic()`.)

We assume that coefficient denominators are "small"; the algorithm factors the denominators, to give the smallest possible scale factor.

EXAMPLES:

```
sage: from sage.rings.qqbar import clear_denominators

sage: _.<x> = QQ['x']
sage: clear_denominators(x + 3/2)
(2, x + 3)
sage: clear_denominators(x^2 + x/2 + 1/4)
(2, x^2 + x + 1)
```

sage.rings.qqbar.**conjugate_expand**(*v*)

If the interval `v` (which may be real or complex) includes some purely real numbers, return `v'` containing `v` such that `v' == v'.conjugate()`. Otherwise return `v` unchanged. (Note that if `v' == v'.conjugate()`, and `v'` includes one non-real root of a real polynomial, then `v'` also includes the conjugate of that root. Also note that the diameter of the return value is at most twice the diameter of the input.)

EXAMPLES:

```
sage: from sage.rings.qqbar import conjugate_expand
sage: conjugate_expand(CIF(RIF(0, 1), RIF(1, 2))).str(style='brackets')
'[0.00000000000000000 .. 1.0000000000000000] + [1.0000000000000000 .. 2.0000000000000000]*I'
sage: conjugate_expand(CIF(RIF(0, 1), RIF(0, 1))).str(style='brackets')
'[0.00000000000000000 .. 1.0000000000000000] + [-1.0000000000000000 .. 1.0000000000000000]*I'
sage: conjugate_expand(CIF(RIF(0, 1), RIF(-2, 1))).str(style='brackets')
'[0.00000000000000000 .. 1.0000000000000000] + [-2.0000000000000000 .. 2.0000000000000000]*I'
sage: conjugate_expand(RIF(1, 2)).str(style='brackets')
'[1.0000000000000000 .. 2.0000000000000000]'
```

sage.rings.qqbar.**conjugate_shrink**(*v*)

If the interval `v` includes some purely real numbers, return a real interval containing only those real numbers. Otherwise return `v` unchanged.

If `v` includes exactly one root of a real polynomial, and `v` was returned by `conjugate_expand()`, then `conjugate_shrink(v)` still includes that root, and is a `RealIntervalFieldElement` iff the root in question is real.

EXAMPLES:

```
sage: from sage.rings.qqbar import conjugate_shrink
sage: conjugate_shrink(RIF(3, 4)).str(style='brackets')
'[3.0000000000000000 .. 4.0000000000000000]'
sage: conjugate_shrink(CIF(RIF(1, 2), RIF(1, 2))).str(style='brackets')
'[1.0000000000000000 .. 2.0000000000000000] + [1.0000000000000000 .. 2.0000000000000000]*I'
sage: conjugate_shrink(CIF(RIF(1, 2), RIF(0, 1))).str(style='brackets')
'[1.0000000000000000 .. 2.0000000000000000]'
sage: conjugate_shrink(CIF(RIF(1, 2), RIF(-1, 2))).str(style='brackets')
'[1.0000000000000000 .. 2.0000000000000000]'
```

sage.rings.qqbar.**cyclotomic_generator**(*n*)

Return an `AlgebraicGenerator` object corresponding to the generator $e^{2\pi I/n}$ of the n-th cyclotomic field.

EXAMPLE:

```
sage: from sage.rings.qqbar import cyclotomic_generator
sage: g=cyclotomic_generator(7); g
1*e^(2*pi*I*1/7)
sage: type(g)
<class 'sage.rings.qqbar.AlgebraicGenerator'>
```

sage.rings.qqbar.**do_polred**(*poly*)

Find a polynomial of reasonably small discriminant that generates the same number field as `poly`, using the

PARI `polredbest` function.

INPUT:

- `poly` - a monic irreducible polynomial with integer coefficients.

OUTPUT:

A triple (`elt_fwd`, `elt_back`, `new_poly`), where:

- `new_poly` is the new polynomial defining the same number field,
- `elt_fwd` is a polynomial expression for a root of the new polynomial in terms of a root of the original polynomial,
- `elt_back` is a polynomial expression for a root of the original polynomial in terms of a root of the new polynomial.

EXAMPLES:

```
sage: from sage.rings.qqbar import do_polred
sage: R.<x> = QQ['x']
sage: oldpol = x^2 - 5
sage: fwd, back, newpol = do_polred(oldpol)
sage: newpol
x^2 - x - 1
sage: Kold.<a> = NumberField(oldpol)
sage: Knew.<b> = NumberField(newpol)
sage: newpol(fwd(a))
0
sage: oldpol(back(b))
0
sage: do_polred(x^2 - x - 11)
(1/3*x + 1/3, 3*x - 1, x^2 - x - 1)
sage: do_polred(x^3 + 123456)
(1/4*x, 4*x, x^3 + 1929)
```

This shows that trac ticket #13054 has been fixed:

```
sage: do_polred(x^4 - 4294967296*x^2 + 54265257667816538374400)
(1/4*x, 4*x, x^4 - 268435456*x^2 + 211973662764908353025)
```

sage.rings.qqbar.**find_zero_result**(*fn*, *l*)

`l` is a list of some sort. `fn` is a function which maps an element of `l` and a precision into an interval (either real or complex) of that precision, such that for sufficient precision, exactly one element of `l` results in an interval containing 0. Returns that one element of `l`.

EXAMPLES:

```
sage: from sage.rings.qqbar import find_zero_result
sage: _.<x> = QQ['x']
sage: delta = 10^(-70)
sage: p1 = x - 1
sage: p2 = x - 1 - delta
sage: p3 = x - 1 + delta
sage: p2 == find_zero_result(lambda p, prec: p(RealIntervalField(prec)(1 + delta)), [p1, p2, p3]
True
```

sage.rings.qqbar.**get_AA_golden_ratio**()

Return the golden ratio as an element of the algebraic real field. Used by `sage.symbolic.constants.golden_ratio._algebraic_()`.

EXAMPLE:

```
sage: AA(golden_ratio) # indirect doctest
1.618033988749895?
```

sage.rings.qqbar.**is_AlgebraicField**(*F*)

Check whether F is an AlgebraicField instance.

EXAMPLE:

```
sage: from sage.rings.qqbar import is_AlgebraicField
sage: [is_AlgebraicField(x) for x in [AA, QQbar, None, 0, "spam"]]
[False, True, False, False, False]
```

sage.rings.qqbar.**is_AlgebraicField_common**(*F*)

Check whether F is an AlgebraicField_common instance.

EXAMPLE:

```
sage: from sage.rings.qqbar import is_AlgebraicField_common
sage: [is_AlgebraicField_common(x) for x in [AA, QQbar, None, 0, "spam"]]
[True, True, False, False, False]
```

sage.rings.qqbar.**is_AlgebraicNumber**(*x*)

Test if x is an instance of AlgebraicNumber. For internal use.

EXAMPLE:

```
sage: from sage.rings.qqbar import is_AlgebraicNumber
sage: is_AlgebraicNumber(AA(sqrt(2)))
False
sage: is_AlgebraicNumber(QQbar(sqrt(2)))
True
sage: is_AlgebraicNumber("spam")
False
```

sage.rings.qqbar.**is_AlgebraicReal**(*x*)

Test if x is an instance of AlgebraicReal. For internal use.

EXAMPLE:

```
sage: from sage.rings.qqbar import is_AlgebraicReal
sage: is_AlgebraicReal(AA(sqrt(2)))
True
sage: is_AlgebraicReal(QQbar(sqrt(2)))
False
sage: is_AlgebraicReal("spam")
False
```

sage.rings.qqbar.**is_AlgebraicRealField**(*F*)

Check whether F is an AlgebraicRealField instance. For internal use.

EXAMPLE:

```
sage: from sage.rings.qqbar import is_AlgebraicRealField
sage: [is_AlgebraicRealField(x) for x in [AA, QQbar, None, 0, "spam"]]
[True, False, False, False, False]
```

sage.rings.qqbar.**isolating_interval**(*intv_fn*, *pol*)

intv_fn is a function that takes a precision and returns an interval of that precision containing some particular root of pol. (It must return better approximations as the precision increases.) pol is an irreducible polynomial with rational coefficients.

Returns an interval containing at most one root of pol.

EXAMPLES:

```
sage: from sage.rings.qqbar import isolating_interval

sage: _.<x> = QQ['x']
sage: isolating_interval(lambda prec: sqrt(RealIntervalField(prec)(2)), x^2 - 2)
1.4142135623730950488?
```

And an example that requires more precision:

```
sage: delta = 10^(-70)
sage: p = (x - 1) * (x - 1 - delta) * (x - 1 + delta)
sage: isolating_interval(lambda prec: RealIntervalField(prec)(1 + delta), p)
1.000000000000000000000000000000000000000000000000000000000000000000000000000000000100000000000000000000000000
```

The function also works with complex intervals and complex roots:

```
sage: p = x^2 - x + 13/36
sage: isolating_interval(lambda prec: ComplexIntervalField(prec)(1/2, 1/3), p)
0.500000000000000000000? + 0.333333333333333334?*I
```

sage.rings.qqbar.**number_field_elements_from_algebraics**(*numbers*, *minimal=False*)

Given a sequence of elements of either AA or QQbar (or a mixture), computes a number field containing all of these elements, these elements as members of that number field, and a homomorphism from the number field back to AA or QQbar.

This may not return the smallest such number field, unless minimal=True is specified.

Also, a single number can be passed, rather than a sequence; and any values which are not elements of AA or QQbar will automatically be coerced to QQbar

This function may be useful for efficiency reasons: doing exact computations in the corresponding number field will be faster than doing exact computations directly in AA or QQbar.

EXAMPLES:

We can use this to compute the splitting field of a polynomial. (Unfortunately this takes an unreasonably long time for non-toy examples.):

```
sage: x = polygen(QQ)
sage: p = x^3 + x^2 + x + 17
sage: rts = p.roots(ring=QQbar, multiplicities=False)
sage: splitting = number_field_elements_from_algebraics(rts)[0]; splitting
Number Field in a with defining polynomial y^6 - 40*y^4 - 22*y^3 + 873*y^2 + 1386*y + 594
sage: p.roots(ring=splitting)
[(361/29286*a^5 - 19/3254*a^4 - 14359/29286*a^3 + 401/29286*a^2 + 18183/1627*a + 15930/1627, 1),
sage: rt2 = AA(sqrt(2)); rt2
1.414213562373095?
sage: rt3 = AA(sqrt(3)); rt3
1.732050807568878?
sage: qqI = QQbar.zeta(4); qqI
1*I
sage: z3 = QQbar.zeta(3); z3
-0.500000000000000? + 0.866025403784439?*I
sage: rt2b = rt3 + rt2 - rt3; rt2b
1.414213562373095?
sage: rt2c = z3 + rt2 - z3; rt2c
1.414213562373095? + 0.?e-18*I

sage: number_field_elements_from_algebraics(rt2)
```

```
(Number Field in a with defining polynomial y^2 - 2, a, Ring morphism:
    From: Number Field in a with defining polynomial y^2 - 2
    To:   Algebraic Real Field
    Defn: a |--> 1.414213562373095?)

sage: number_field_elements_from_algebraics((rt2,rt3))
(Number Field in a with defining polynomial y^4 - 4*y^2 + 1, [-a^3 + 3*a, -a^2 + 2], Ring morphi
    From: Number Field in a with defining polynomial y^4 - 4*y^2 + 1
    To:   Algebraic Real Field
    Defn: a |--> 0.5176380902050415?)
```

We've created `rt2b` in such a way that sage doesn't initially know that it's in a degree-2 extension of $\mathbf{Q}$:

```
sage: number_field_elements_from_algebraics(rt2b)
(Number Field in a with defining polynomial y^4 - 4*y^2 + 1, -a^3 + 3*a, Ring morphism:
    From: Number Field in a with defining polynomial y^4 - 4*y^2 + 1
    To:   Algebraic Real Field
    Defn: a |--> 0.5176380902050415?)
```

We can specify `minimal=True` if we want the smallest number field:

```
sage: number_field_elements_from_algebraics(rt2b, minimal=True)
(Number Field in a with defining polynomial y^2 - 2, a, Ring morphism:
    From: Number Field in a with defining polynomial y^2 - 2
    To:   Algebraic Real Field
    Defn: a |--> 1.414213562373095?)
```

Things work fine with rational numbers, too:

```
sage: number_field_elements_from_algebraics((QQbar(1/2), AA(17)))
(Rational Field, [1/2, 17], Ring morphism:
    From: Rational Field
    To:   Algebraic Real Field
    Defn: 1 |--> 1)
```

Or we can just pass in symbolic expressions, as long as they can be coerced into `QQbar`:

```
sage: number_field_elements_from_algebraics((sqrt(7), sqrt(9), sqrt(11)))
(Number Field in a with defining polynomial y^4 - 9*y^2 + 1, [-a^3 + 8*a, 3, -a^3 + 10*a], Ring
    From: Number Field in a with defining polynomial y^4 - 9*y^2 + 1
    To:   Algebraic Real Field
    Defn: a |--> 0.3354367396454047?)
```

Here we see an example of doing some computations with number field elements, and then mapping them back into `QQbar`:

```
sage: (fld,nums,hom) = number_field_elements_from_algebraics((rt2, rt3, qqI, z3))
sage: fld,nums,hom  # random
(Number Field in a with defining polynomial y^8 - y^4 + 1, [-a^5 + a^3 + a, a^6 - 2*a^2, a^6, -a
  From: Number Field in a with defining polynomial y^8 - y^4 + 1
  To:   Algebraic Field
  Defn: a |--> -0.2588190451025208? - 0.9659258262890683?*I)
sage: (nfrt2, nfrt3, nfI, nfz3) = nums
sage: hom(nfrt2)
1.414213562373095? + 0.?e-18*I
sage: nfrt2^2
2
sage: nfrt3^2
3
```

```
sage: nfz3 + nfz3^2
-1
sage: nfI^2
-1
sage: sum = nfrt2 + nfrt3 + nfI + nfz3; sum
-a^5 + a^4 + a^3 - 2*a^2 + a - 1
sage: hom(sum)
2.646264369941973? + 1.866025403784439?*I
sage: hom(sum) == rt2 + rt3 + qqI + z3
True
sage: [hom(n) for n in nums] == [rt2, rt3, qqI, z3]
True
```

TESTS:

```
sage: number_field_elements_from_algebraics(rt3)
(Number Field in a with defining polynomial y^2 - 3, a, Ring morphism:
    From: Number Field in a with defining polynomial y^2 - 3
    To:   Algebraic Real Field
    Defn: a |--> 1.732050807568878?)
sage: number_field_elements_from_algebraics((rt2,qqI))
(Number Field in a with defining polynomial y^4 + 1, [a^3 - a, -a^2], Ring morphism:
    From: Number Field in a with defining polynomial y^4 + 1
    To:   Algebraic Field
    Defn: a |--> -0.7071067811865475? + 0.7071067811865475?*I)
```

Note that for the first example, where sage doesn't realize that the number is real, we get a homomorphism to `QQbar`; but with `minimal=True`, we get a homomorphism to `AA`. Also note that the exact answer depends on a Pari function that gives different answers for 32-bit and 64-bit machines:

```
sage: number_field_elements_from_algebraics(rt2c)
(Number Field in a with defining polynomial y^4 + 2*y^2 + 4, 1/2*a^3, Ring morphism:
    From: Number Field in a with defining polynomial y^4 + 2*y^2 + 4
    To:   Algebraic Field
    Defn: a |--> -0.7071067811865475? - 1.224744871391589?*I)
sage: number_field_elements_from_algebraics(rt2c, minimal=True)
(Number Field in a with defining polynomial y^2 - 2, a, Ring morphism:
    From: Number Field in a with defining polynomial y^2 - 2
    To:   Algebraic Real Field
    Defn: a |--> 1.414213562373095?)
```

sage.rings.qqbar.**prec_seq**()

Return a generator object which iterates over an infinite increasing sequence of precisions to be tried in various numerical computations.

Currently just returns powers of 2 starting at 64.

EXAMPLE:

```
sage: g = sage.rings.qqbar.prec_seq()
sage: [next(g), next(g), next(g)]
[64, 128, 256]
```

sage.rings.qqbar.**rational_exact_root**(r, d)

Checks whether the rational r is an exact d'th power. If so, returns the d'th root of r; otherwise, returns None.

EXAMPLES:

```
sage: from sage.rings.qqbar import rational_exact_root
sage: rational_exact_root(16/81, 4)
```

```
2/3
sage: rational_exact_root(8/81, 3) is None
True
```

sage.rings.qqbar.**short_prec_seq**()

Return a sequence of precisions to try in cases when an infinite-precision computation is possible: returns a couple of small powers of 2 and then None.

EXAMPLE:

```
sage: from sage.rings.qqbar import short_prec_seq
sage: short_prec_seq()
(64, 128, None)
```

sage.rings.qqbar.**tail_prec_seq**()

A generator over precisions larger than those in short_prec_seq().

EXAMPLE:

```
sage: from sage.rings.qqbar import tail_prec_seq
sage: g = tail_prec_seq()
sage: [next(g), next(g), next(g)]
[256, 512, 1024]
```

CHAPTER

TWENTYFOUR

UNIVERSAL CYCLOTOMIC FIELD

The universal cyclotomic field is the smallest subfield of the complex field containing all roots of unity. It is also the maximal Galois Abelian extension of the rational numbers.

The implementation simply wraps GAP Cyclotomic. As mentioned in their documentation: arithmetical operations are quite expensive, so the use of internally represented cyclotomics is not recommended for doing arithmetic over number fields, such as calculations with matrices of cyclotomics.

Note: There used to be a native Sage version of the universal cyclotomic field written by Christian Stump (see trac ticket #8327). It was slower on most operations and it was decided to use a version based on libGAP instead (see trac ticket #18152). One main difference in the design choices is that GAP stores dense vectors whereas the native ones used Python dictionaries (storing only nonzero coefficients). Most operations are faster with libGAP except some operation on very sparse elements. All details can be found in trac ticket #18152.

REFERENCES:

EXAMPLES:

```
sage: UCF = UniversalCyclotomicField(); UCF
Universal Cyclotomic Field
```

To generate cyclotomic elements:

```
sage: UCF.gen(5)
E(5)
sage: UCF.gen(5,2)
E(5)^2

sage: E = UCF.gen
```

Equality and inequality checks:

```
sage: E(6,2) == E(6)^2 == E(3)
True

sage: E(6)^2 != E(3)
False
```

Addition and multiplication:

```
sage: E(2) * E(3)
-E(3)
sage: f = E(2) + E(3); f
2*E(3) + E(3)^2
```

Inverses:

```
sage: f^-1
1/3*E(3) + 2/3*E(3)^2
sage: f.inverse()
1/3*E(3) + 2/3*E(3)^2
sage: f * f.inverse()
1
```

Conjugation and Galois conjugates:

```
sage: f.conjugate()
E(3) + 2*E(3)^2

sage: f.galois_conjugates()
[2*E(3) + E(3)^2, E(3) + 2*E(3)^2]
sage: f.norm_of_galois_extension()
3
```

One can create matrices and polynomials:

```
sage: m = matrix(2,[E(3),1,1,E(4)]); m
[E(3)    1]
[   1 E(4)]
sage: m.parent()
Full MatrixSpace of 2 by 2 dense matrices over Universal Cyclotomic Field
sage: m**2
[                      -E(3) E(12)^4 - E(12)^7 - E(12)^11]
[E(12)^4 - E(12)^7 - E(12)^11                           0]

sage: m.charpoly()
x^2 + (-E(12)^4 + E(12)^7 + E(12)^11)*x + E(12)^4 + E(12)^7 + E(12)^8

sage: m.echelon_form()
[1 0]
[0 1]

sage: m.pivots()
(0, 1)

sage: m.rank()
2

sage: R.<x> = PolynomialRing(UniversalCyclotomicField(), 'x')
sage: E(3) * x - 1
E(3)*x - 1
```

TESTS:

```
sage: UCF.one()
1
sage: UCF.zero()
0
sage: UCF.one().is_one()
True
sage: UCF.one().is_zero()
False
sage: UCF.zero().is_zero()
True
```

Check that trac ticket #14240 is fixed:

```
sage: K.<rho> = CyclotomicField(245)
sage: h = K.random_element()
sage: h_rho = rho.coordinates_in_terms_of_powers()(h)
sage: h_ucf = sum( c * E(245, i) for (i, c) in enumerate(h_rho) )
sage: h_ucf**2  # random
-169539876343/589714020*E(245) + 27815735177/20058300*E(245)^2  + ... + + 7828432097501/842448600*E(2
```

Check that trac ticket #16130 is fixed:

```
sage: mat = matrix(UCF, 2, [-4, 2*E(7)^6, -5*E(13)^3 + 5*E(13)^8 - 4*E(13)^9, 0])
sage: mat._echelon_classical()
[1 0]
[0 1]
```

Check that trac ticket #16631 is fixed:

```
sage: UCF.one() / 2
1/2
sage: UCF.one() / 2r
1/2
```

Check that trac ticket #17117 is fixed:

```
sage: e3 = UCF.gen(3)
sage: N(e3)
-0.500000000000000 + 0.866025403784439*I
sage: real(e3)
-1/2
sage: imag(e3)
-1/2*E(12)^7 + 1/2*E(12)^11
```

AUTHORS:

- Christian Stump (2013): initial Sage version (see trac ticket #8327)
- Vincent Delecroix (2015): complete rewriting using libgap (see trac ticket #18152)

sage.rings.universal_cyclotomic_field.**E**(*n*, *k=1*)

Return the n-th root of unity as an element of the universal cyclotomic field.

EXAMPLES:

```
sage: E(3)
E(3)
sage: E(3) + E(5)
-E(15)^2 - 2*E(15)^8 - E(15)^11 - E(15)^13 - E(15)^14
```

class sage.rings.universal_cyclotomic_field.**UCFtoQQbar**(*UCF*)

Bases: sage.categories.morphism.Morphism

Conversion to QQbar.

EXAMPLES:

```
sage: UCF = UniversalCyclotomicField()
sage: QQbar(UCF.gen(3))
-0.500000000000000? + 0.866025403784439?*I

sage: CC(UCF.gen(7,2) + UCF.gen(7,6))
0.400968867902419 + 0.193096429713794*I
```

```
sage: complex(E(7)+E(7,2))
(0.40096886790241915+1.7567593946498534j)
sage: complex(UCF.one()/2)
(0.5+0j)
```

class sage.rings.universal_cyclotomic_field.**UniversalCyclotomicField**(*names=None*)

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

The universal cyclotomic field.

The universal cyclotomic field is the infinite algebraic extension of $\mathbf{Q}$ generated by the roots of unity. It is also the maximal Abelian extension of $\mathbf{Q}$ in the sense that any Abelian Galois extension of $\mathbf{Q}$ is also a subfield of the universal cyclotomic field.

Element

alias of UniversalCyclotomicFieldElement

algebraic_closure()

The algebraic closure.

EXAMPLES:

```
sage: UniversalCyclotomicField().algebraic_closure()
Algebraic Field
```

an_element()

Return an element.

EXAMPLES:

```
sage: UniversalCyclotomicField().an_element()
E(5) - 3*E(5)^2
```

characteristic()

Return the characteristic.

EXAMPLES:

```
sage: UniversalCyclotomicField().characteristic()
0
sage: parent(_)
Integer Ring
```

degree()

Returns the *degree* of self as a field extension over the Rationals.

EXAMPLES:

```
sage: UCF = UniversalCyclotomicField()
sage: UCF.degree()
+Infinity
```

gen(*n*, *k=1*)

Return the standard n-th root of unity.

If k is not None, return the k-th power of it.

EXAMPLES:

```
sage: UCF = UniversalCyclotomicField()
sage: UCF.gen(15)
```

```
E(15)
sage: UCF.gen(7,3)
E(7)^3
sage: UCF.gen(4,2)
-1
```

is_exact()
Return `True` as this is an exact ring (i.e. not numerical).

EXAMPLES:

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

is_finite()
Returns `True`.

EXAMPLES:

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

Todo

this method should be provided by the category.

---

one()
Return one.

EXAMPLES:

```
sage: UCF = UniversalCyclotomicField()
sage: UCF.one()
1
sage: parent(_)
Universal Cyclotomic Field
```

some_elements()
Return a tuple of some elements in the universal cyclotomic field.

EXAMPLES:

```
sage: UniversalCyclotomicField().some_elements()
(0, 1, -1, E(3), E(7) - 2/3*E(7)^2)
sage: all(parent(x) is UniversalCyclotomicField() for x in _)
True
```

zero()
Return zero.

EXAMPLES:

```
sage: UCF = UniversalCyclotomicField()
sage: UCF.zero()
0
sage: parent(_)
Universal Cyclotomic Field
```

class sage.rings.universal_cyclotomic_field.**UniversalCyclotomicFieldElement**(*parent*, *obj*)

Bases: sage.structure.element.FieldElement

INPUT:

- `parent` - a universal cyclotomic field
- `obj` - a libgap element (either an integer, a rational or a cyclotomic)

TESTS:

```
sage: UCF = UniversalCyclotomicField()
sage: a = UCF.an_element()
sage: TestSuite(a).run()
```

additive_order()

The additive order.

EXAMPLES:

```
sage: UCF = UniversalCyclotomicField()
sage: UCF.zero().additive_order()
0
sage: UCF.one().additive_order()
+Infinity
sage: UCF.gen(3).additive_order()
+Infinity
```

conductor()

Return the conductor of `self`.

EXAMPLES:

```
sage: E(3).conductor()
3
sage: (E(5) + E(3)).conductor()
15
```

conjugate()

Return the complex conjugate.

EXAMPLES:

```
sage: (E(7) + 3*E(7,2) - 5 * E(7,3)).conjugate()
-5*E(7)^4 + 3*E(7)^5 + E(7)^6
```

denominator()

Return the denominator of this element.

EXAMPLES:

```
sage: a = E(5) + 1/2*E(5,2) + 1/3*E(5,3)
sage: a
E(5) + 1/2*E(5)^2 + 1/3*E(5)^3
sage: a.denominator()
6
sage: parent(_)
Integer Ring
```

field_order(**args*, ***kwds*)

Deprecated: Use `conductor()` instead. See trac ticket #18152 for details.

galois_conjugates(*n=None*)

Return the Galois conjugates of `self`.

INPUT:

- n – an optional integer. If provided, return the orbit of the Galois group of the n-th cyclotomic field over $\mathbf{Q}$. Note that n must be such that this element belongs to the n-th cyclotomic field (in other words, it must be a multiple of the conductor).

EXAMPLES:

```
sage: E(6).galois_conjugates()
[-E(3)^2, -E(3)]

sage: E(6).galois_conjugates()
[-E(3)^2, -E(3)]

sage: (E(9,2) - E(9,4)).galois_conjugates()
[E(9)^2 - E(9)^4,
 E(9)^2 + E(9)^4 + E(9)^5,
 -E(9)^2 - E(9)^5 - E(9)^7,
 -E(9)^2 - E(9)^4 - E(9)^7,
 E(9)^4 + E(9)^5 + E(9)^7,
 -E(9)^5 + E(9)^7]

sage: zeta = E(5)
sage: zeta.galois_conjugates(5)
[E(5), E(5)^2, E(5)^3, E(5)^4]
sage: zeta.galois_conjugates(10)
[E(5), E(5)^3, E(5)^2, E(5)^4]
sage: zeta.galois_conjugates(15)
[E(5), E(5)^2, E(5)^4, E(5)^2, E(5)^3, E(5), E(5)^3, E(5)^4]

sage: zeta.galois_conjugates(17)
Traceback (most recent call last):
...
ValueError: n = 17 must be a multiple of the conductor (5)
```

imag()

Return the imaginary part of this element.

EXAMPLES:

```
sage: E(3).imag()
-1/2*E(12)^7 + 1/2*E(12)^11
sage: E(5).imag()
1/2*E(20) - 1/2*E(20)^9

sage: a = E(5) - 2*E(3)
sage: AA(a.imag()) == QQbar(a).imag()
True
```

imag_part()

Return the imaginary part of this element.

EXAMPLES:

```
sage: E(3).imag()
-1/2*E(12)^7 + 1/2*E(12)^11
sage: E(5).imag()
1/2*E(20) - 1/2*E(20)^9

sage: a = E(5) - 2*E(3)
sage: AA(a.imag()) == QQbar(a).imag()
True
```

inverse()

TESTS:

```
sage: UCF = UniversalCyclotomicField()
sage: ~(UCF.one())
1
sage: ~UCF.gen(4)
-E(4)
```

is_rational()

Test whether this element is a rational number.

EXAMPLES:

```
sage: E(3).is_rational()
False
sage: (E(3) + E(3,2)).is_rational()
True
```

TESTS:

```
sage: type(E(3).is_rational())
<type 'bool'>
```

is_real()

Test whether this element is real.

EXAMPLES:

```
sage: E(3).is_real()
False
sage: (E(3) + E(3,2)).is_real()
True

sage: a = E(3) - 2*E(7)
sage: a.real_part().is_real()
True
sage: a.imag_part().is_real()
True
```

minpoly(*var='x'*)

The minimal polynomial of `self` element over $\mathbf{Q}$.

INPUT:

- `var` – (optional, default 'x') the name of the variable to use.

EXAMPLES:

```
sage: UCF.<E> = UniversalCyclotomicField()

sage: UCF(4).minpoly()
x - 4

sage: UCF(4).minpoly(var='y')
y - 4

sage: E(3).minpoly()
x^2 + x + 1

sage: E(3).minpoly(var='y')
y^2 + y + 1
```

TESTS:

```
sage: for elt in UCF.some_elements():
....:     assert elt.minpoly() == elt.to_cyclotomic_field().minpoly()
....:     assert elt.minpoly(var='y') == elt.to_cyclotomic_field().minpoly(var='y')
```

Todo

Polynomials with libgap currently does not implement a `.sage()` method (see trac ticket #18266). It would be faster/safer to not use string to construct the polynomial.

---

multiplicative_order()

The multiplicative order.

EXAMPLES:

```
sage: E(5).multiplicative_order()
5
sage: (E(5) + E(12)).multiplicative_order()
+Infinity
sage: UniversalCyclotomicField().zero().multiplicative_order()
Traceback (most recent call last):
...
ValueError: libGAP: Error, argument must be nonzero
```

norm_of_galois_extension()

Returns the norm as a Galois extension of $\mathbf{Q}$, which is given by the product of all galois_conjugates.

EXAMPLES:

```
sage: E(3).norm_of_galois_extension()
1
sage: E(6).norm_of_galois_extension()
1
sage: (E(2) + E(3)).norm_of_galois_extension()
3
sage: parent(_)
Integer Ring
```

real()

Return the real part of this element.

EXAMPLES:

```
sage: E(3).real()
-1/2
sage: E(5).real()
1/2*E(5) + 1/2*E(5)^4

sage: a = E(5) - 2*E(3)
sage: AA(a.real()) == QQbar(a).real()
True
```

real_part()

Return the real part of this element.

EXAMPLES:

```
sage: E(3).real()
-1/2
sage: E(5).real()
```

```
1/2*E(5) + 1/2*E(5)^4

sage: a = E(5) - 2*E(3)
sage: AA(a.real()) == QQbar(a).real()
True
```

to_cyclotomic_field(*R=None*)

Return this element as an element of a cyclotomic field.

EXAMPLES:

```
sage: UCF = UniversalCyclotomicField()

sage: UCF.gen(3).to_cyclotomic_field()
zeta3
sage: UCF.gen(3,2).to_cyclotomic_field()
-zeta3 - 1

sage: CF = CyclotomicField(5)
sage: CF(E(5)) # indirect doctest
zeta5

sage: CF = CyclotomicField(7)
sage: CF(E(5)) # indirect doctest
Traceback (most recent call last):
...
TypeError: Cannot coerce zeta5 into Cyclotomic Field of order 7 and
degree 6

sage: CF = CyclotomicField(10)
sage: CF(E(5)) # indirect doctest
zeta10^2
```

Matrices are correctly dealt with:

```
sage: M = Matrix(UCF,2,[E(3),E(4),E(5),E(6)]); M
[   E(3)    E(4)]
[   E(5) -E(3)^2]

sage: Matrix(CyclotomicField(60),M) # indirect doctest
[zeta60^10 - 1     zeta60^15]
[    zeta60^12     zeta60^10]
```

Using a non-standard embedding:

```
sage: CF = CyclotomicField(5,embedding=CC(exp(4*pi*i/5)))
sage: x = E(5)
sage: CC(x)
0.309016994374947 + 0.951056516295154*I
sage: CC(CF(x))
0.309016994374947 + 0.951056516295154*I
```

sage.rings.universal_cyclotomic_field.**late_import**()

This function avoids importing libgap on startup. It is called once through the constrcturo of UniversalCyclotomicField.

EXAMPLES:

```
sage: import sage.rings.universal_cyclotomic_field as ucf
sage: _ = UniversalCyclotomicField()   # indirect doctest
```

```
sage: ucf.libgap is None                # indirect doctest
False
```

CHAPTER

TWENTYFIVE

INDICES AND TABLES

- Index
- Module Index
- Search Page

[C] H. Cohen. A Course in Computational Algebraic Number Theory. Springer-Verlag, 1993.

[Doyle-Krumm] John R. Doyle and David Krumm, Computing algebraic numbers of bounded height, Arxiv 1111.4963 (2013).

[Cohen2000] Henri Cohen, Advanced topics in computational number theory, Graduate Texts in Mathematics, vol. 193, Springer-Verlag, New York, 2000.

[Martinet1980] Jacques Martinet, Petits discriminants des corps de nombres, Journ. Arithm. 1980, Cambridge Univ. Press, 1982, 151–193.

[Takeuchi1999] Kisao Takeuchi, Totally real algebraic number fields of degree 9 with small discriminant, Saitama Math. J. 17 (1999), 63–85 (2000).

[Voight2008] John Voight, Enumeration of totally real number fields of bounded root discriminant, Lect. Notes in Comp. Sci. 5011 (2008).

[CE] Henry Cohn and Noam Elkies, New upper bounds on sphere packings I, Ann. Math. 157 (2003), 689–714.

[CS] J.H. Conway and N.J.A. Sloane, Sphere packings, lattices and groups, 3rd. ed., Grundlehren der Mathematischen Wissenschaften, vol. 290, Springer-Verlag, New York, 1999.

[Bre97] 20. Breuer "Integral bases for subfields of cyclotomic fields" AAECC 8, 279–289 (1997).

r

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W

Z