
Algebraic Function Fields

Release 10.6

The Sage Development Team

Jun 27, 2025

CONTENTS

1	Function Fields	3
2	Function Fields: rational	41
3	Function Fields: extension	59
4	Elements of function fields	105
5	Elements of function fields: rational	127
6	Elements of function fields: extension	135
7	Orders of function fields	139
8	Orders of function fields: rational	145
9	Orders of function fields: basis	151
10	Orders of function fields: extension	165
11	Ideals of function fields	183
12	Ideals of function fields: rational	205
13	Ideals of function fields: extension	213
14	Places of function fields	237
15	Places of function fields: rational	243
16	Places of function fields: extension	247
17	Divisors of function fields	253
18	Differentials of function fields	265
19	Valuation rings of function fields	279
20	Derivations of function fields	283
21	Derivations of function fields: rational	285
22	Derivations of function fields: extension	287

23 Morphisms of function fields	293
24 Special extensions of function fields	305
25 Factories to construct function fields	311
26 Jacobians of function fields	315
27 A Support Module	351
28 Indices and Tables	355
Python Module Index	357
Index	359

Sage allows basic computations with elements and ideals in orders of algebraic function fields over arbitrary constant fields. Advanced computations, like computing the genus or a basis of the Riemann-Roch space of a divisor, are available for function fields over finite fields, number fields, and the algebraic closure of $\mathbf{Q}$.

FUNCTION FIELDS

A function field (of one variable) is a finitely generated field extension of transcendence degree one. In Sage, a function field can be a rational function field or a finite extension of a function field.

EXAMPLES:

We create a rational function field:

```
sage: # needs sage.rings.finite_rings
sage: K.<x> = FunctionField(GF(5^2,'a')); K
Rational function field in x over Finite Field in a of size 5^2
sage: K.genus()
0
sage: f = (x^2 + x + 1) / (x^3 + 1)
sage: f
(x^2 + x + 1)/(x^3 + 1)
sage: f^3
(x^6 + 3*x^5 + x^4 + 2*x^3 + x^2 + 3*x + 1)/(x^9 + 3*x^6 + 3*x^3 + 1)
```

```
>>> from sage.all import *
>>> # needs sage.rings.finite_rings
>>> K = FunctionField(GF(Integer(5)**Integer(2),'a'), names=('x',)); (x,) = K._first_
↳ngens(1); K
Rational function field in x over Finite Field in a of size 5^2
>>> K.genus()
0
>>> f = (x**Integer(2) + x + Integer(1)) / (x**Integer(3) + Integer(1))
>>> f
(x^2 + x + 1)/(x^3 + 1)
>>> f**Integer(3)
(x^6 + 3*x^5 + x^4 + 2*x^3 + x^2 + 3*x + 1)/(x^9 + 3*x^6 + 3*x^3 + 1)
```

Then we create an extension of the rational function field, and do some simple arithmetic in it:

```
sage: # needs sage.rings.finite_rings sage.rings.function_field
sage: R.<y> = K[]
sage: L.<y> = K.extension(y^3 - (x^3 + 2*x*y + 1/x)); L
Function field in y defined by y^3 + 3*x*y + (4*x^4 + 4)/x
sage: y^2
y^2
sage: y^3
2*x*y + (x^4 + 1)/x
```

(continues on next page)

(continued from previous page)

```
sage: a = 1/y; a
(x/(x^4 + 1))*y^2 + 3*x^2/(x^4 + 1)
sage: a * y
1
```

```
>>> from sage.all import *
>>> # needs sage.rings.finite_rings sage.rings.function_field
>>> R = K['y']; (y,) = R._first_ngens(1)
>>> L = K.extension(y**Integer(3) - (x**Integer(3) + Integer(2)*x*y + Integer(1)/x),
↳names=('y',)); (y,) = L._first_ngens(1); L
Function field in y defined by y^3 + 3*x*y + (4*x^4 + 4)/x
>>> y**Integer(2)
y^2
>>> y**Integer(3)
2*x*y + (x^4 + 1)/x
>>> a = Integer(1)/y; a
(x/(x^4 + 1))*y^2 + 3*x^2/(x^4 + 1)
>>> a * y
1
```

We next make an extension of the above function field, illustrating that arithmetic with a tower of three fields is fully supported:

```
sage: # needs sage.rings.finite_rings sage.rings.function_field
sage: S.<t> = L[]
sage: M.<t> = L.extension(t^2 - x*y)
sage: M
Function field in t defined by t^2 + 4*x*y
sage: t^2
x*y
sage: 1/t
((1/(x^4 + 1))*y^2 + 3*x/(x^4 + 1))*t
sage: M.base_field()
Function field in y defined by y^3 + 3*x*y + (4*x^4 + 4)/x
sage: M.base_field().base_field()
Rational function field in x over Finite Field in a of size 5^2
```

```
>>> from sage.all import *
>>> # needs sage.rings.finite_rings sage.rings.function_field
>>> S = L['t']; (t,) = S._first_ngens(1)
>>> M = L.extension(t**Integer(2) - x*y, names=('t',)); (t,) = M._first_ngens(1)
>>> M
Function field in t defined by t^2 + 4*x*y
>>> t**Integer(2)
x*y
>>> Integer(1)/t
((1/(x^4 + 1))*y^2 + 3*x/(x^4 + 1))*t
>>> M.base_field()
Function field in y defined by y^3 + 3*x*y + (4*x^4 + 4)/x
>>> M.base_field().base_field()
Rational function field in x over Finite Field in a of size 5^2
```


It is also possible to construct function fields over an imperfect base field:

```
sage: N.<u> = FunctionField(K)
↳needs sage.rings.finite_rings
```

```
>>> from sage.all import *
>>> N = FunctionField(K, names=('u',)); (u,) = N._first_ngens(1) # needs sage.rings.
↳finite_rings
```

and inseparable extension function fields:

```
sage: J.<x> = FunctionField(GF(5)); J
Rational function field in x over Finite Field of size 5
sage: T.<v> = J[]
sage: O.<v> = J.extension(v^5 - x); O
↳needs sage.rings.function_field
Function field in v defined by v^5 + 4*x
```

```
>>> from sage.all import *
>>> J = FunctionField(GF(Integer(5)), names=('x',)); (x,) = J._first_ngens(1); J
Rational function field in x over Finite Field of size 5
>>> T = J['v']; (v,) = T._first_ngens(1)
>>> O = J.extension(v**Integer(5) - x, names=('v',)); (v,) = O._first_ngens(1); O
↳ # needs sage.rings.function_field
Function field in v defined by v^5 + 4*x
```

Function fields over the rational field are supported:

```
sage: # needs sage.rings.function_field
sage: F.<x> = FunctionField(QQ)
sage: R.<Y> = F[]
sage: L.<y> = F.extension(Y^2 - x^8 - 1)
sage: O = L.maximal_order()
sage: I = O.ideal(x, y - 1)
sage: P = I.place()
sage: D = P.divisor()
sage: D.basis_function_space()
[1]
sage: (2*D).basis_function_space()
[1]
sage: (3*D).basis_function_space()
[1]
sage: (4*D).basis_function_space()
[1, 1/x^4*y + 1/x^4]

sage: # needs sage.rings.function_field
sage: K.<x> = FunctionField(QQ); _.<Y> = K[]
sage: F.<y> = K.extension(Y^3 - x^2*(x^2 + x + 1)^2)
sage: O = F.maximal_order()
sage: I = O.ideal(y)
sage: I.divisor()
2*Place (x, y, (1/(x^3 + x^2 + x))*y^2)
+ 2*Place (x^2 + x + 1, y, (1/(x^3 + x^2 + x))*y^2)
```

(continues on next page)

(continued from previous page)

```

sage: # needs sage.rings.function_field
sage: K.<x> = FunctionField(QQ); _.<Y> = K[]
sage: L.<y> = K.extension(Y^2 + Y + x + 1/x)
sage: O = L.maximal_order()
sage: I = O.ideal(y)
sage: I.divisor()
- Place (x, x*y)
+ Place (x^2 + 1, x*y)

```

```

>>> from sage.all import *
>>> # needs sage.rings.function_field
>>> F = FunctionField(QQ, names=('x',)); (x,) = F._first_ngens(1)
>>> R = F['Y']; (Y,) = R._first_ngens(1)
>>> L = F.extension(Y**Integer(2) - x**Integer(8) - Integer(1), names=('y',)); (y,) = L._first_ngens(1)
>>> O = L.maximal_order()
>>> I = O.ideal(x, y - Integer(1))
>>> P = I.place()
>>> D = P.divisor()
>>> D.basis_function_space()
[1]
>>> (Integer(2)*D).basis_function_space()
[1]
>>> (Integer(3)*D).basis_function_space()
[1]
>>> (Integer(4)*D).basis_function_space()
[1, 1/x^4*y + 1/x^4]

>>> # needs sage.rings.function_field
>>> K = FunctionField(QQ, names=('x',)); (x,) = K._first_ngens(1); _ = K['Y']; (Y,) = _._first_ngens(1)
>>> F = K.extension(Y**Integer(3) - x**Integer(2)*(x**Integer(2) + x + Integer(1))**Integer(2), names=('y',)); (y,) = F._first_ngens(1)
>>> O = F.maximal_order()
>>> I = O.ideal(y)
>>> I.divisor()
2*Place (x, y, (1/(x^3 + x^2 + x))*y^2)
+ 2*Place (x^2 + x + 1, y, (1/(x^3 + x^2 + x))*y^2)

>>> # needs sage.rings.function_field
>>> K = FunctionField(QQ, names=('x',)); (x,) = K._first_ngens(1); _ = K['Y']; (Y,) = _._first_ngens(1)
>>> L = K.extension(Y**Integer(2) + Y + x + Integer(1)/x, names=('y',)); (y,) = L._first_ngens(1)
>>> O = L.maximal_order()
>>> I = O.ideal(y)
>>> I.divisor()
- Place (x, x*y)
+ Place (x^2 + 1, x*y)

```

Function fields over the algebraic field are supported:

```

sage: # needs sage.rings.function_field sage.rings.number_field
sage: K.<x> = FunctionField(QQbar); _.<Y> = K[]
sage: L.<y> = K.extension(Y^2 + Y + x + 1/x)
sage: O = L.maximal_order()
sage: I = O.ideal(y)
sage: I.divisor()
Place (x - I, x*y)
- Place (x, x*y)
+ Place (x + I, x*y)
sage: p1 = I.divisor().support()[0]
sage: m = L.completion(p1, prec=5)
sage: m(x)
I + s + O(s^5)
sage: m(y)
# long time (4s)
-2*s + (-4 - I)*s^2 + (-15 - 4*I)*s^3 + (-75 - 23*I)*s^4 + (-413 - 154*I)*s^5 + O(s^6)
sage: m(y)^2 + m(y) + m(x) + 1/m(x)
# long time (8s)
O(s^5)

```

```

>>> from sage.all import *
>>> # needs sage.rings.function_field sage.rings.number_field
>>> K = FunctionField(QQbar, names=('x',)); (x,) = K._first_ngens(1); _ = K['Y']; (Y,
↪) = _._first_ngens(1)
>>> L = K.extension(Y**Integer(2) + Y + x + Integer(1)/x, names=('y',)); (y,) = L._
↪first_ngens(1)
>>> O = L.maximal_order()
>>> I = O.ideal(y)
>>> I.divisor()
Place (x - I, x*y)
- Place (x, x*y)
+ Place (x + I, x*y)
>>> p1 = I.divisor().support()[Integer(0)]
>>> m = L.completion(p1, prec=Integer(5))
>>> m(x)
I + s + O(s^5)
>>> m(y)
# long time (4s)
-2*s + (-4 - I)*s^2 + (-15 - 4*I)*s^3 + (-75 - 23*I)*s^4 + (-413 - 154*I)*s^5 + O(s^6)
>>> m(y)**Integer(2) + m(y) + m(x) + Integer(1)/m(x)
# long time (8s)
O(s^5)

```

1.1 Global function fields

A global function field in Sage is an extension field of a rational function field over a *finite* constant field by an irreducible separable polynomial over the rational function field.

EXAMPLES:

A fundamental computation for a global or any function field is to get a basis of its maximal order and maximal infinite order, and then do arithmetic with ideals of those maximal orders:

```

sage: # needs sage.rings.function_field
sage: K.<x> = FunctionField(GF(3)); _.<t> = K[]
sage: L.<y> = K.extension(t^4 + t - x^5)

```

(continues on next page)

(continued from previous page)

```

sage: O = L.maximal_order()
sage: O.basis()
(1, y, 1/x*y^2 + 1/x*y, 1/x^3*y^3 + 2/x^3*y^2 + 1/x^3*y)
sage: I = O.ideal(x,y); I
Ideal (x, y) of Maximal order of Function field in y defined by y^4 + y + 2*x^5
sage: J = I^-1
sage: J.basis_matrix()
[ 1  0  0  0]
[1/x 1/x  0  0]
[ 0  0  1  0]
[ 0  0  0  1]
sage: L.maximal_order_infinite().basis()
(1, 1/x^2*y, 1/x^3*y^2, 1/x^4*y^3)

```

```

>>> from sage.all import *
>>> # needs sage.rings.function_field
>>> K = FunctionField(GF(Integer(3)), names=('x',)); (x,) = K._first_ngens(1); _ = K[
↳ 't']; (t,) = _._first_ngens(1)
>>> L = K.extension(t**Integer(4) + t - x**Integer(5), names=('y',)); (y,) = L._first_
↳ ngens(1)
>>> O = L.maximal_order()
>>> O.basis()
(1, y, 1/x*y^2 + 1/x*y, 1/x^3*y^3 + 2/x^3*y^2 + 1/x^3*y)
>>> I = O.ideal(x,y); I
Ideal (x, y) of Maximal order of Function field in y defined by y^4 + y + 2*x^5
>>> J = I**Integer(-1)
>>> J.basis_matrix()
[ 1  0  0  0]
[1/x 1/x  0  0]
[ 0  0  1  0]
[ 0  0  0  1]
>>> L.maximal_order_infinite().basis()
(1, 1/x^2*y, 1/x^3*y^2, 1/x^4*y^3)

```

As an example of the most sophisticated computations that Sage can do with a global function field, we compute all the Weierstrass places of the Klein quartic over $\mathbf{F}_2$ and gap numbers for ordinary places:

```

sage: # needs sage.rings.function_field
sage: K.<x> = FunctionField(GF(2)); _.<Y> = K[]
sage: L.<y> = K.extension(Y^3 + x^3*Y + x)
sage: L.genus()
3
sage: L.weierstrass_places() #_
↳ needs sage.modules
[Place (1/x, 1/x^3*y^2 + 1/x),
 Place (1/x, 1/x^3*y^2 + 1/x^2*y + 1),
 Place (x, y),
 Place (x + 1, (x^3 + 1)*y + x + 1),
 Place (x^3 + x + 1, y + 1),
 Place (x^3 + x + 1, y + x^2),
 Place (x^3 + x + 1, y + x^2 + 1),
 Place (x^3 + x^2 + 1, y + x),

```

(continues on next page)

(continued from previous page)

```

Place (x^3 + x^2 + 1, y + x^2 + 1),
Place (x^3 + x^2 + 1, y + x^2 + x + 1)]
sage: L.gaps()
↳needs sage.modules
[1, 2, 3]

```

```

>>> from sage.all import *
>>> # needs sage.rings.function_field
>>> K = FunctionField(GF(Integer(2)), names=('x',)); (x,) = K._first_ngens(1); _ = K[
↳'Y']; (Y,) = _._first_ngens(1)
>>> L = K.extension(Y**Integer(3) + x**Integer(3)*Y + x, names=('y',)); (y,) = L._
↳first_ngens(1)
>>> L.genus()
3
>>> L.weierstrass_places()
↳needs sage.modules
[Place (1/x, 1/x^3*y^2 + 1/x),
Place (1/x, 1/x^3*y^2 + 1/x^2*y + 1),
Place (x, y),
Place (x + 1, (x^3 + 1)*y + x + 1),
Place (x^3 + x + 1, y + 1),
Place (x^3 + x + 1, y + x^2),
Place (x^3 + x + 1, y + x^2 + 1),
Place (x^3 + x^2 + 1, y + x),
Place (x^3 + x^2 + 1, y + x^2 + 1),
Place (x^3 + x^2 + 1, y + x^2 + x + 1)]
>>> L.gaps()
↳needs sage.modules
[1, 2, 3]

```

The gap numbers for Weierstrass places are of course not ordinary:

```

sage: # needs sage.rings.function_field
sage: p1,p2,p3 = L.weierstrass_places()[:3]
sage: p1.gaps()
[1, 2, 4]
sage: p2.gaps()
[1, 2, 4]
sage: p3.gaps()
[1, 2, 4]

```

```

>>> from sage.all import *
>>> # needs sage.rings.function_field
>>> p1,p2,p3 = L.weierstrass_places()[:Integer(3)]
>>> p1.gaps()
[1, 2, 4]
>>> p2.gaps()
[1, 2, 4]
>>> p3.gaps()
[1, 2, 4]

```

AUTHORS:

- William Stein (2010): initial version
- Robert Bradshaw (2010-05-30): added `is_finite()`
- Julian R  th (2011-06-08, 2011-09-14, 2014-06-23, 2014-06-24, 2016-11-13): fixed `hom()`, `extension()`; use `@cached_method`; added `derivation()`; added support for relative vector spaces; fixed conversion to base fields
- Maarten Derickx (2011-09-11): added `doctests`
- Syed Ahmad Lavasani (2011-12-16): added `genus()`, `is_RationalFunctionField()`
- Simon King (2014-10-29): Use the same generator names for a function field extension and the underlying polynomial ring.
- Kwankyu Lee (2017-04-30): added global function fields
- Brent Baccala (2019-12-20): added function fields over number fields and $\overline{\mathbb{Q}\mathbb{Q}}$
- Sebastian A. Spindler (2024-03-06): implemented Hilbert symbols for global function fields

```
class sage.rings.function_field.function_field.FunctionField(base_field, names,
                                                             category=Category of function
                                                             fields)
```

Bases: `Field`

Abstract base class for all function fields.

INPUT:

- `base_field` – field; the base of this function field
- `names` – string that gives the name of the generator

EXAMPLES:

```
sage: K.<x> = FunctionField(QQ)
sage: K
Rational function field in x over Rational Field
```

```
>>> from sage.all import *
>>> K = FunctionField(QQ, names=('x',)); (x,) = K._first_ngens(1)
>>> K
Rational function field in x over Rational Field
```

`basis_of_differentials_of_first_kind()`

Return a basis of the space of holomorphic differentials of this function field.

EXAMPLES:

```
sage: K.<t> = FunctionField(QQ)
sage: K.basis_of_holomorphic_differentials() #_
↳needs sage.libs.pari sage.modules
[]

sage: K.<x> = FunctionField(GF(5)); _.<Y> = K[]
sage: L.<y> = K.extension(Y^3 - (x^3 - 1)/(x^3 - 2)) #_
↳needs sage.rings.function_field
sage: L.basis_of_holomorphic_differentials() #_
↳needs sage.rings.function_field
[(x/(x^3 + 4))*y d(x),
```

(continues on next page)

(continued from previous page)

```
((1/(x^3 + 4))*y) d(x),
((x/(x^3 + 4))*y^2) d(x),
((1/(x^3 + 4))*y^2) d(x)]
```

```
>>> from sage.all import *
>>> K = FunctionField(QQ, names=('t',)); (t,) = K._first_ngens(1)
>>> K.basis_of_holomorphic_differentials() #_
↳needs sage.libs.pari sage.modules
[]

>>> K = FunctionField(GF(Integer(5)), names=('x',)); (x,) = K._first_ngens(1);
↳ _ = K['Y']; (Y,) = _._first_ngens(1)
>>> L = K.extension(Y**Integer(3) - (x**Integer(3) - Integer(1))/
↳ (x**Integer(3) - Integer(2)), names=('y',)); (y,) = L._first_ngens(1) #_
↳needs sage.rings.function_field
>>> L.basis_of_holomorphic_differentials() #_
↳needs sage.rings.function_field
[((x/(x^3 + 4))*y) d(x),
 ((1/(x^3 + 4))*y) d(x),
 ((x/(x^3 + 4))*y^2) d(x),
 ((1/(x^3 + 4))*y^2) d(x)]
```

basis_of_holomorphic_differentials()

Return a basis of the space of holomorphic differentials of this function field.

EXAMPLES:

```
sage: K.<t> = FunctionField(QQ)
sage: K.basis_of_holomorphic_differentials() #_
↳needs sage.libs.pari sage.modules
[]

sage: K.<x> = FunctionField(GF(5)); _.<Y> = K[]
sage: L.<y> = K.extension(Y^3 - (x^3 - 1)/(x^3 - 2)) #_
↳needs sage.rings.function_field
sage: L.basis_of_holomorphic_differentials() #_
↳needs sage.rings.function_field
[((x/(x^3 + 4))*y) d(x),
 ((1/(x^3 + 4))*y) d(x),
 ((x/(x^3 + 4))*y^2) d(x),
 ((1/(x^3 + 4))*y^2) d(x)]
```

```
>>> from sage.all import *
>>> K = FunctionField(QQ, names=('t',)); (t,) = K._first_ngens(1)
>>> K.basis_of_holomorphic_differentials() #_
↳needs sage.libs.pari sage.modules
[]

>>> K = FunctionField(GF(Integer(5)), names=('x',)); (x,) = K._first_ngens(1);
↳ _ = K['Y']; (Y,) = _._first_ngens(1)
>>> L = K.extension(Y**Integer(3) - (x**Integer(3) - Integer(1))/
↳ (x**Integer(3) - Integer(2)), names=('y',)); (y,) = L._first_ngens(1) #_
```

(continues on next page)

(continued from previous page)

```

↪needs sage.rings.function_field
>>> L.basis_of_holomorphic_differentials() #_
↪needs sage.rings.function_field
[(x/(x^3 + 4))*y d(x),
 (1/(x^3 + 4))*y d(x),
 (x/(x^3 + 4))*y^2 d(x),
 (1/(x^3 + 4))*y^2 d(x)]

```

characteristic()

Return the characteristic of the function field.

EXAMPLES:

```

sage: K.<x> = FunctionField(QQ)
sage: K.characteristic()
0
sage: K.<x> = FunctionField(QQbar) #_
↪needs sage.rings.number_field
sage: K.characteristic()
0
sage: K.<x> = FunctionField(GF(7))
sage: K.characteristic()
7
sage: R.<y> = K[]
sage: L.<y> = K.extension(y^2 - x) #_
↪needs sage.rings.function_field
sage: L.characteristic() #_
↪needs sage.rings.function_field
7

```

```

>>> from sage.all import *
>>> K = FunctionField(QQ, names=('x',)); (x,) = K._first_ngens(1)
>>> K.characteristic()
0
>>> K = FunctionField(QQbar, names=('x',)); (x,) = K._first_ngens(1) # needs_
↪sage.rings.number_field
>>> K.characteristic()
0
>>> K = FunctionField(GF(Integer(7)), names=('x',)); (x,) = K._first_ngens(1)
>>> K.characteristic()
7
>>> R = K['y']; (y,) = R._first_ngens(1)
>>> L = K.extension(y**Integer(2) - x, names=('y',)); (y,) = L._first_ngens(1)
↪# needs sage.rings.function_field
>>> L.characteristic() #_
↪needs sage.rings.function_field
7

```

completion (*place*, *name=None*, *prec=None*, *gen_name=None*)

Return the completion of the function field at the place.

INPUT:

- *place* – place

- name – string; name of the series variable
- prec – positive integer; default precision
- gen_name – string; name of the generator of the residue field; used only when the place is non-rational

EXAMPLES:

```

sage: # needs sage.rings.function_field
sage: K.<x> = FunctionField(GF(2)); _.<Y> = K[]
sage: L.<y> = K.extension(Y^2 + Y + x + 1/x)
sage: p = L.places_finite()[0]
sage: m = L.completion(p); m
Completion map:
  From: Function field in y defined by y^2 + y + (x^2 + 1)/x
  To:   Laurent Series Ring in s over Finite Field of size 2
sage: m(x, 10)
s^2 + s^3 + s^4 + s^5 + s^7 + s^8 + s^9 + s^10 + O(s^12)
sage: m(y, 10)
s^-1 + 1 + s^3 + s^5 + s^7 + O(s^9)

sage: # needs sage.rings.function_field
sage: K.<x> = FunctionField(GF(2)); _.<Y> = K[]
sage: L.<y> = K.extension(Y^2 + Y + x + 1/x)
sage: p = L.places_finite()[0]
sage: m = L.completion(p); m
Completion map:
  From: Function field in y defined by y^2 + y + (x^2 + 1)/x
  To:   Laurent Series Ring in s over Finite Field of size 2
sage: m(x, 10)
s^2 + s^3 + s^4 + s^5 + s^7 + s^8 + s^9 + s^10 + O(s^12)
sage: m(y, 10)
s^-1 + 1 + s^3 + s^5 + s^7 + O(s^9)

sage: K.<x> = FunctionField(GF(2))
sage: p = K.places_finite()[0]; p                                     #_
↪needs sage.libs.pari
Place (x)
sage: m = K.completion(p); m                                         #_
↪needs sage.rings.function_field
Completion map:
  From: Rational function field in x over Finite Field of size 2
  To:   Laurent Series Ring in s over Finite Field of size 2
sage: m(1/(x+1))                                                     #_
↪needs sage.rings.function_field
1 + s + s^2 + s^3 + s^4 + s^5 + s^6 + s^7 + s^8 + s^9 + s^10 + s^11 + s^12
+ s^13 + s^14 + s^15 + s^16 + s^17 + s^18 + s^19 + O(s^20)

sage: p = K.place_infinite(); p
Place (1/x)
sage: m = K.completion(p); m                                         #_
↪needs sage.rings.function_field
Completion map:
  From: Rational function field in x over Finite Field of size 2
  To:   Laurent Series Ring in s over Finite Field of size 2

```

(continues on next page)

(continued from previous page)

```

sage: m(x) #_
↪needs sage.rings.function_field
s^-1 + O(s^19)

sage: m = K.completion(p, prec=infinity); m #_
↪needs sage.rings.function_field
Completion map:
  From: Rational function field in x over Finite Field of size 2
  To:   Lazy Laurent Series Ring in s over Finite Field of size 2

sage: f = m(x); f #_
↪needs sage.rings.function_field
s^-1 + ...

sage: f.coefficient(100) #_
↪needs sage.rings.function_field
0

sage: # needs sage.rings.function_field
sage: K.<x> = FunctionField(QQ); _.<Y> = K[]
sage: L.<y> = K.extension(Y^2 - x)
sage: O = L.maximal_order()
sage: decomp = O.decomposition(K.maximal_order().ideal(x - 1))
sage: pls = (decomp[0][0].place(), decomp[1][0].place())
sage: m = L.completion(pls[0]); m
Completion map:
  From: Function field in y defined by y^2 - x
  To:   Laurent Series Ring in s over Rational Field

sage: xe = m(x)
sage: ye = m(y)
sage: ye^2 - xe == 0
True

sage: # needs sage.rings.function_field
sage: decomp2 = O.decomposition(K.maximal_order().ideal(x^2 + 1))
sage: pls2 = decomp2[0][0].place()
sage: m = L.completion(pls2); m
Completion map:
  From: Function field in y defined by y^2 - x
  To:   Laurent Series Ring in s over
        Number Field in a with defining polynomial x^4 + 2*x^2 + 4*x + 2

sage: xe = m(x)
sage: ye = m(y)
sage: ye^2 - xe == 0
True

>>> from sage.all import *
>>> # needs sage.rings.function_field
>>> K = FunctionField(GF(Integer(2)), names=('x',)); (x,) = K._first_ngens(1);
↪ _ = K['Y']; (Y,) = _._first_ngens(1)
>>> L = K.extension(Y**Integer(2) + Y + x + Integer(1)/x, names=('y',)); (y,) #_
↪ L._first_ngens(1)
>>> p = L.places_finite()[Integer(0)]
>>> m = L.completion(p); m

```

(continues on next page)

(continued from previous page)

```

Completion map:
  From: Function field in y defined by  $y^2 + y + (x^2 + 1)/x$ 
  To:   Laurent Series Ring in s over Finite Field of size 2
>>> m(x, Integer(10))
 $s^2 + s^3 + s^4 + s^5 + s^7 + s^8 + s^9 + s^{10} + O(s^{12})$ 
>>> m(y, Integer(10))
 $s^{-1} + 1 + s^3 + s^5 + s^7 + O(s^9)$ 

>>> # needs sage.rings.function_field
>>> K = FunctionField(GF(Integer(2)), names=('x',)); (x,) = K._first_ngens(1);
↪ _ = K['Y']; (Y,) = _._first_ngens(1)
>>> L = K.extension(Y**Integer(2) + Y + x + Integer(1)/x, names=('y',)); (y,) =
↪ L._first_ngens(1)
>>> p = L.places_finite()[Integer(0)]
>>> m = L.completion(p); m
Completion map:
  From: Function field in y defined by  $y^2 + y + (x^2 + 1)/x$ 
  To:   Laurent Series Ring in s over Finite Field of size 2
>>> m(x, Integer(10))
 $s^2 + s^3 + s^4 + s^5 + s^7 + s^8 + s^9 + s^{10} + O(s^{12})$ 
>>> m(y, Integer(10))
 $s^{-1} + 1 + s^3 + s^5 + s^7 + O(s^9)$ 

>>> K = FunctionField(GF(Integer(2)), names=('x',)); (x,) = K._first_ngens(1)
>>> p = K.places_finite()[Integer(0)]; p
↪ # needs sage.libs.pari
Place (x)
>>> m = K.completion(p); m
↪ # needs sage.rings.function_field
Completion map:
  From: Rational function field in x over Finite Field of size 2
  To:   Laurent Series Ring in s over Finite Field of size 2
>>> m(Integer(1)/(x+Integer(1)))
↪ # needs sage.rings.function_field
 $1 + s + s^2 + s^3 + s^4 + s^5 + s^6 + s^7 + s^8 + s^9 + s^{10} + s^{11} + s^{12} + s^{13} + s^{14} + s^{15} + s^{16} + s^{17} + s^{18} + s^{19} + O(s^{20})$ 

>>> p = K.place_infinite(); p
Place (1/x)
>>> m = K.completion(p); m
↪ # needs sage.rings.function_field
Completion map:
  From: Rational function field in x over Finite Field of size 2
  To:   Laurent Series Ring in s over Finite Field of size 2
>>> m(x)
↪ # needs sage.rings.function_field
 $s^{-1} + O(s^{19})$ 

>>> m = K.completion(p, prec=infinity); m
↪ # needs sage.rings.function_field
Completion map:
  From: Rational function field in x over Finite Field of size 2

```

(continues on next page)

(continued from previous page)

```

To:    Lazy Laurent Series Ring in s over Finite Field of size 2
>>> f = m(x); f                                     #_
↪needs sage.rings.function_field
s^-1 + ...
>>> f.coefficient(Integer(100))                      _
↪    # needs sage.rings.function_field
0

>>> # needs sage.rings.function_field
>>> K = FunctionField(QQ, names=('x',)); (x,) = K._first_ngens(1); _ = K['Y'];
↪ (Y,) = _._first_ngens(1)
>>> L = K.extension(Y**Integer(2) - x, names=('y',)); (y,) = L._first_ngens(1)
>>> O = L.maximal_order()
>>> decomp = O.decomposition(K.maximal_order().ideal(x - Integer(1)))
>>> pls = (decomp[Integer(0)][Integer(0)].place(),_
↪ decomp[Integer(1)][Integer(0)].place())
>>> m = L.completion(pls[Integer(0)]); m
Completion map:
  From: Function field in y defined by y^2 - x
  To:    Laurent Series Ring in s over Rational Field
>>> xe = m(x)
>>> ye = m(y)
>>> ye**Integer(2) - xe == Integer(0)
True

>>> # needs sage.rings.function_field
>>> decomp2 = O.decomposition(K.maximal_order().ideal(x**Integer(2) +_
↪ Integer(1)))
>>> pls2 = decomp2[Integer(0)][Integer(0)].place()
>>> m = L.completion(pls2); m
Completion map:
  From: Function field in y defined by y^2 - x
  To:    Laurent Series Ring in s over
          Number Field in a with defining polynomial x^4 + 2*x^2 + 4*x + 2
>>> xe = m(x)
>>> ye = m(y)
>>> ye**Integer(2) - xe == Integer(0)
True

```

divisor_group()

Return the group of divisors attached to the function field.

EXAMPLES:

```

sage: K.<t> = FunctionField(QQ)
sage: K.divisor_group()                             #_
↪needs sage.modules
Divisor group of Rational function field in t over Rational Field

sage: _.<Y> = K[]
sage: L.<y> = K.extension(Y^3 - (t^3 - 1)/(t^3 - 2))   #_
↪needs sage.rings.function_field

```

(continues on next page)

(continued from previous page)

```

sage: L.divisor_group() #_
↪needs sage.rings.function_field
Divisor group of Function field in y defined by  $y^3 + (-t^3 + 1)/(t^3 - 2)$ 

sage: K.<x> = FunctionField(GF(5)); _.<Y> = K[]
sage: L.<y> = K.extension(Y^3 - (x^3 - 1)/(x^3 - 2)) #_
↪needs sage.rings.function_field
sage: L.divisor_group() #_
↪needs sage.rings.function_field
Divisor group of Function field in y defined by  $y^3 + (4*x^3 + 1)/(x^3 + 3)$ 

```

```

>>> from sage.all import *
>>> K = FunctionField(QQ, names=('t',)); (t,) = K._first_ngens(1)
>>> K.divisor_group() #_
↪needs sage.modules
Divisor group of Rational function field in t over Rational Field

>>> _ = K['Y']; (Y,) = _._first_ngens(1)
>>> L = K.extension(Y**Integer(3) - (t**Integer(3) - Integer(1))/
↪(t**Integer(3) - Integer(2)), names=('y',)); (y,) = L._first_ngens(1) #_
↪needs sage.rings.function_field
>>> L.divisor_group() #_
↪needs sage.rings.function_field
Divisor group of Function field in y defined by  $y^3 + (-t^3 + 1)/(t^3 - 2)$ 

>>> K = FunctionField(GF(Integer(5)), names=('x',)); (x,) = K._first_ngens(1);
↪_ = K['Y']; (Y,) = _._first_ngens(1)
>>> L = K.extension(Y**Integer(3) - (x**Integer(3) - Integer(1))/
↪(x**Integer(3) - Integer(2)), names=('y',)); (y,) = L._first_ngens(1) #_
↪needs sage.rings.function_field
>>> L.divisor_group() #_
↪needs sage.rings.function_field
Divisor group of Function field in y defined by  $y^3 + (4*x^3 + 1)/(x^3 + 3)$ 

```

extension (*f*, names=None)

Create an extension $K(y)$ of this function field K extended with a root y of the univariate polynomial f over K .

INPUT:

- f – univariate polynomial over K
- names – string or tuple of length 1 that names the variable y

OUTPUT: a function field

EXAMPLES:

```

sage: K.<x> = FunctionField(QQ); R.<y> = K[]
sage: K.extension(y^5 - x^3 - 3*x + x*y) #_
↪needs sage.rings.function_field
Function field in y defined by  $y^5 + x*y - x^3 - 3*x$ 

```

```
>>> from sage.all import *
>>> K = FunctionField(QQ, names=('x',)); (x,) = K._first_ngens(1); R = K['y'];
↪ (y,) = R._first_ngens(1)
>>> K.extension(y**Integer(5) - x**Integer(3) - Integer(3)*x + x*y)
↪ # needs sage.rings.function_field
Function field in y defined by  $y^5 + x^3y - x^3 - 3x$ 
```

A nonintegral defining polynomial:

```
sage: K.<t> = FunctionField(QQ); R.<y> = K[]
sage: K.extension(y^3 + (1/t)*y + t^3/(t+1), 'z')
↪ # needs sage.rings.function_field
Function field in z defined by  $z^3 + 1/tz + t^3/(t + 1)$ 
```

```
>>> from sage.all import *
>>> K = FunctionField(QQ, names=('t',)); (t,) = K._first_ngens(1); R = K['y'];
↪ (y,) = R._first_ngens(1)
>>> K.extension(y**Integer(3) + (Integer(1)/t)*y + t**Integer(3)/
↪ (t+Integer(1)), 'z')
↪ # needs sage.rings.function_
↪ field
Function field in z defined by  $z^3 + 1/tz + t^3/(t + 1)$ 
```

The defining polynomial need not be monic or integral:

```
sage: K.extension(t*y^3 + (1/t)*y + t^3/(t+1))
↪ # needs sage.rings.function_field
Function field in y defined by  $t^3y^3 + 1/t^2y + t^3/(t + 1)$ 
```

```
>>> from sage.all import *
>>> K.extension(t*y**Integer(3) + (Integer(1)/t)*y + t**Integer(3)/
↪ (t+Integer(1)))
↪ # needs sage.rings.function_
↪ field
Function field in y defined by  $t^3y^3 + 1/t^2y + t^3/(t + 1)$ 
```

`extension_constant_field(k)`

Return the constant field extension with constant field k .

INPUT:

- k – an extension field of the constant field of this function field

EXAMPLES:

```
sage: # needs sage.rings.function_field
sage: K.<x> = FunctionField(GF(2)); _.<Y> = K[]
sage: F.<y> = K.extension(Y^2 + Y + x + 1/x)
sage: E = F.extension_constant_field(GF(2^4))
sage: E
Function field in y defined by  $y^2 + y + (x^2 + 1)/x$  over its base
sage: E.constant_base_field()
Finite Field in z4 of size  $2^4$ 
```

```
>>> from sage.all import *
>>> # needs sage.rings.function_field
```

(continues on next page)

(continued from previous page)

```

>>> K = FunctionField(GF(Integer(2)), names=('x',)); (x,) = K._first_ngens(1);
↪ _ = K['Y']; (Y,) = _._first_ngens(1)
>>> F = K.extension(Y**Integer(2) + Y + x + Integer(1)/x, names=('y',)); (y,) =
↪ F._first_ngens(1)
>>> E = F.extension_constant_field(GF(Integer(2)**Integer(4)))
>>> E
Function field in y defined by y^2 + y + (x^2 + 1)/x over its base
>>> E.constant_base_field()
Finite Field in z4 of size 2^4

```

hilbert_symbol(a, b, P)

Return the Hilbert symbol $(a, b)_{F_P}$ for the local field F_P .

The local field F_P is the completion of this function field F at the place P .

INPUT:

- a, b – elements of this function field
- P – a place of this function field

The Hilbert symbol $(a, b)_{F_P}$ is 0 if a or b is zero. Otherwise it takes the value 1 if the quaternion algebra defined by (a, b) over F_P is split, and -1 if said algebra is a division ring.

ALGORITHM:

For the valuation $\nu = \nu_P$ of F , we compute the valuations $\nu(a)$ and $\nu(b)$ as well as elements a_0 and b_0 of the residue field such that for a uniformizer π at P , $a\pi^{-\nu(a)}$ respectively $b\pi^{-\nu(b)}$ has the residue class a_0 respectively b_0 modulo π . Then the Hilbert symbol is computed by formula 12.4.10 in [Voi2021].

Currently only implemented for global function fields.

EXAMPLES:

```

sage: K.<x> = FunctionField(GF(17))
sage: P = K.places()[0]; P
Place (1/x)
sage: a = (5*x + 6)/(x + 15)
sage: b = 7/x
sage: K.hilbert_symbol(a, b, P)
-1

sage: Q = K.places()[7]; Q
Place (x + 6)
sage: c = 15*x + 12
sage: d = 16/(x + 13)
sage: K.hilbert_symbol(c, d, Q)
1

```

```

>>> from sage.all import *
>>> K = FunctionField(GF(Integer(17)), names=('x',)); (x,) = K._first_ngens(1)
>>> P = K.places()[Integer(0)]; P
Place (1/x)
>>> a = (Integer(5)*x + Integer(6))/(x + Integer(15))
>>> b = Integer(7)/x
>>> K.hilbert_symbol(a, b, P)

```

(continues on next page)

(continued from previous page)

```

-1

>>> Q = K.places()[Integer(7)]; Q
Place (x + 6)
>>> c = Integer(15)*x + Integer(12)
>>> d = Integer(16)/(x + Integer(13))
>>> K.hilbert_symbol(c, d, Q)
1

```

Check that the Hilbert symbol is symmetric and bimultiplicative:

```

sage: K.<x> = FunctionField(GF(5)); R.<T> = PolynomialRing(K)
sage: f = ((x^2 + 2*x + 2)*T^5 + (4*x^2 + 2*x + 3)*T^4 + 3*T^3 + 4*T^2
....:      + (2/(x^2 + 4*x + 1))*T + 3*x^2 + 2*x + 4)
sage: L.<y> = K.extension(f)
sage: a = L.random_element()
sage: b = L.random_element()
sage: c = L.random_element()
sage: P = L.places_above(K.places()[0])[1]
sage: Q = L.places_above(K.places()[1])[0]

sage: hP_a_c = L.hilbert_symbol(a, c, P)
sage: hP_a_c == L.hilbert_symbol(c, a, P)
True
sage: L.hilbert_symbol(a, b, P) * hP_a_c == L.hilbert_symbol(a, b*c, P)
True
sage: hP_a_c * L.hilbert_symbol(b, c, P) == L.hilbert_symbol(a*b, c, P)
True

sage: hQ_a_c = L.hilbert_symbol(a, c, Q)
sage: hQ_a_c == L.hilbert_symbol(c, a, Q)
True
sage: L.hilbert_symbol(a, b, Q) * hQ_a_c == L.hilbert_symbol(a, b*c, Q)
True
sage: hQ_a_c * L.hilbert_symbol(b, c, Q) == L.hilbert_symbol(a*b, c, Q)
True

```

```

>>> from sage.all import *
>>> K = FunctionField(GF(Integer(5)), names=('x',)); (x,) = K._first_ngens(1);
↳ R = PolynomialRing(K, names=('T',)); (T,) = R._first_ngens(1)
>>> f = ((x**Integer(2) + Integer(2)*x + Integer(2))*T**Integer(5) +
↳ (Integer(4)*x**Integer(2) + Integer(2)*x + Integer(3))*T**Integer(4) +
↳ Integer(3)*T**Integer(3) + Integer(4)*T**Integer(2)
...      + (Integer(2)/(x**Integer(2) + Integer(4)*x + Integer(1)))*T +
↳ Integer(3)*x**Integer(2) + Integer(2)*x + Integer(4))
>>> L = K.extension(f, names=('y',)); (y,) = L._first_ngens(1)
>>> a = L.random_element()
>>> b = L.random_element()
>>> c = L.random_element()
>>> P = L.places_above(K.places()[Integer(0)])[Integer(1)]
>>> Q = L.places_above(K.places()[Integer(1)])[Integer(0)]

```

(continues on next page)

(continued from previous page)

```

>>> hP_a_c = L.hilbert_symbol(a, c, P)
>>> hP_a_c == L.hilbert_symbol(c, a, P)
True
>>> L.hilbert_symbol(a, b, P) * hP_a_c == L.hilbert_symbol(a, b*c, P)
True
>>> hP_a_c * L.hilbert_symbol(b, c, P) == L.hilbert_symbol(a*b, c, P)
True

>>> hQ_a_c = L.hilbert_symbol(a, c, Q)
>>> hQ_a_c == L.hilbert_symbol(c, a, Q)
True
>>> L.hilbert_symbol(a, b, Q) * hQ_a_c == L.hilbert_symbol(a, b*c, Q)
True
>>> hQ_a_c * L.hilbert_symbol(b, c, Q) == L.hilbert_symbol(a*b, c, Q)
True

```

is_finite()

Return whether the function field is finite, which is false.

EXAMPLES:

```

sage: R.<t> = FunctionField(QQ)
sage: R.is_finite()
False
sage: R.<t> = FunctionField(GF(7))
sage: R.is_finite()
False

```

```

>>> from sage.all import *
>>> R = FunctionField(QQ, names=('t',)); (t,) = R._first_ngens(1)
>>> R.is_finite()
False
>>> R = FunctionField(GF(Integer(7)), names=('t',)); (t,) = R._first_ngens(1)
>>> R.is_finite()
False

```

is_global()

Return whether the function field is global, that is, whether the constant field is finite.

EXAMPLES:

```

sage: R.<t> = FunctionField(QQ)
sage: R.is_global()
False
sage: R.<t> = FunctionField(QQbar)
↪needs sage.rings.number_field
sage: R.is_global()
False
sage: R.<t> = FunctionField(GF(7))
sage: R.is_global()
True

```

```

>>> from sage.all import *
>>> R = FunctionField(QQ, names=('t',)); (t,) = R._first_ngens(1)
>>> R.is_global()
False
>>> R = FunctionField(QQbar, names=('t',)); (t,) = R._first_ngens(1) # needs_
↪ sage.rings.number_field
>>> R.is_global()
False
>>> R = FunctionField(GF(Integer(7)), names=('t',)); (t,) = R._first_ngens(1)
>>> R.is_global()
True

```

is_perfect()

Return whether the field is perfect, i.e., its characteristic p is zero or every element has a p -th root.

EXAMPLES:

```

sage: FunctionField(QQ, 'x').is_perfect()
True
sage: FunctionField(GF(2), 'x').is_perfect()
False

```

```

>>> from sage.all import *
>>> FunctionField(QQ, 'x').is_perfect()
True
>>> FunctionField(GF(Integer(2)), 'x').is_perfect()
False

```

jacobian (*model=None, base_div=None, **kwds*)

Return the Jacobian of the function field.

INPUT:

- *model* – (default: 'hess') model to use for arithmetic
- *base_div* – an effective divisor

The degree of the base divisor should satisfy certain degree condition corresponding to the model used. The following table lists these conditions. Let g be the genus of the function field.

- *hess*: ideal-based arithmetic; requires base divisor of degree g
- *km_large*: Khuri-Makdisi's large model; requires base divisor of degree at least $2g + 1$
- *km_medium*: Khuri-Makdisi's medium model; requires base divisor of degree at least $2g + 1$
- *km_small*: Khuri-Makdisi's small model requires base divisor of degree at least $g + 1$

We assume the function field has a rational place. If a base divisor is not given, one is constructed using an arbitrary rational place.

EXAMPLES:

```

sage: A.<x,y> = AffineSpace(GF(5), 2)
sage: C = Curve(y^2*(x^3 - 1) - (x^3 - 2))
sage: F = C.function_field()
sage: F.jacobian()

```

(continues on next page)

(continued from previous page)

```
Jacobian of Function field in y defined by (x^3 + 4)*y^2 + 4*x^3 + 2 (Hess
↪model)
```

```
>>> from sage.all import *
>>> A = AffineSpace(GF(Integer(5)), Integer(2), names=('x', 'y',)); (x, y,) =
↪A._first_ngens(2)
>>> C = Curve(y**Integer(2)*(x**Integer(3) - Integer(1)) - (x**Integer(3) -
↪Integer(2)))
>>> F = C.function_field()
>>> F.jacobian()
Jacobian of Function field in y defined by (x^3 + 4)*y^2 + 4*x^3 + 2 (Hess
↪model)
```

order (*x*, *check=True*)

Return the order generated by *x* over the base maximal order.

INPUT:

- *x* – element or list of elements of the function field
- *check* – boolean; if True, check that *x* really generates an order

EXAMPLES:

```
sage: # needs sage.rings.function_field
sage: K.<x> = FunctionField(QQ); R.<y> = K[]
sage: L.<y> = K.extension(y^3 + x^3 + 4*x + 1)
sage: O = L.order(y); O                                     #_
↪needs sage.modules
Order in Function field in y defined by y^3 + x^3 + 4*x + 1
sage: O.basis()                                             #_
↪needs sage.modules
(1, y, y^2)

sage: Z = K.order(x); Z                                     #_
↪needs sage.rings.function_field
Order in Rational function field in x over Rational Field
sage: Z.basis()                                             #_
↪needs sage.rings.function_field
(1,)
```

```
>>> from sage.all import *
>>> # needs sage.rings.function_field
>>> K = FunctionField(QQ, names=('x',)); (x,) = K._first_ngens(1); R = K['y'];
↪(y,) = R._first_ngens(1)
>>> L = K.extension(y**Integer(3) + x**Integer(3) + Integer(4)*x + Integer(1),
↪names=('y',)); (y,) = L._first_ngens(1)
>>> O = L.order(y); O                                     #_
↪needs sage.modules
Order in Function field in y defined by y^3 + x^3 + 4*x + 1
>>> O.basis()                                             #_
↪needs sage.modules
(1, y, y^2)
```

(continues on next page)

(continued from previous page)

```

>>> Z = K.order(x); Z                                     #_
↳needs sage.rings.function_field
Order in Rational function field in x over Rational Field
>>> Z.basis()                                             #_
↳needs sage.rings.function_field
(1,)

```

Orders with multiple generators are not yet supported:

```

sage: Z = K.order([x, x^2]); Z                             #_
↳needs sage.rings.function_field
Traceback (most recent call last):
...
NotImplementedError

```

```

>>> from sage.all import *
>>> Z = K.order([x, x**Integer(2)]); Z                     #_
↳      # needs sage.rings.function_field
Traceback (most recent call last):
...
NotImplementedError

```

order_infinite(*x*, *check=True*)

Return the order generated by *x* over the maximal infinite order.

INPUT:

- *x* – element or a list of elements of the function field
- *check* – boolean; if True, check that *x* really generates an order

EXAMPLES:

```

sage: K.<x> = FunctionField(QQ); R.<y> = K[]
sage: L.<y> = K.extension(y^3 + x^3 + 4*x + 1)               #_
↳needs sage.rings.function_field
sage: L.order_infinite(y)                                  # not implemented #_
↳needs sage.rings.function_field

sage: Z = K.order(x); Z                                    #_
↳needs sage.modules
Order in Rational function field in x over Rational Field
sage: Z.basis()                                           #_
↳needs sage.modules
(1,)

```

```

>>> from sage.all import *
>>> K = FunctionField(QQ, names=('x',)); (x,) = K._first_ngens(1); R = K['y'];
↳ (y,) = R._first_ngens(1)
>>> L = K.extension(y**Integer(3) + x**Integer(3) + Integer(4)*x + Integer(1),
↳ names=('y',)); (y,) = L._first_ngens(1) # needs sage.rings.function_field
>>> L.order_infinite(y)                                    # not implemented #_
↳needs sage.rings.function_field

```

(continues on next page)

(continued from previous page)

```

>>> Z = K.order(x); Z                                     #_
↳needs sage.modules
Order in Rational function field in x over Rational Field
>>> Z.basis()                                             #_
↳needs sage.modules
(1,)

```

Orders with multiple generators, not yet supported:

```

sage: Z = K.order_infinite([x, x^2]); Z
Traceback (most recent call last):
...
NotImplementedError

```

```

>>> from sage.all import *
>>> Z = K.order_infinite([x, x**Integer(2)]); Z
Traceback (most recent call last):
...
NotImplementedError

```

order_infinite_with_basis (*basis*, *check=True*)

Return the order with given basis over the maximal infinite order of the base field.

INPUT:

- *basis* – list of elements of the function field
- *check* – boolean (default: True); if True, check that the basis is really linearly independent and that the module it spans is closed under multiplication, and contains the identity element.

EXAMPLES:

```

sage: # needs sage.rings.function_field
sage: K.<x> = FunctionField(QQ); R.<y> = K[]
sage: L.<y> = K.extension(y^3 + x^3 + 4*x + 1)
sage: O = L.order_infinite_with_basis([1, 1/x*y, 1/x^2*y^2]); O
Infinite order in Function field in y defined by y^3 + x^3 + 4*x + 1
sage: O.basis()
(1, 1/x*y, 1/x^2*y^2)

```

```

>>> from sage.all import *
>>> # needs sage.rings.function_field
>>> K = FunctionField(QQ, names=('x',)); (x,) = K._first_ngens(1); R = K['y'];
↳ (y,) = R._first_ngens(1)
>>> L = K.extension(y**Integer(3) + x**Integer(3) + Integer(4)*x + Integer(1),
↳ names=('y',)); (y,) = L._first_ngens(1)
>>> O = L.order_infinite_with_basis([Integer(1), Integer(1)/x*y, Integer(1)/
↳ x**Integer(2)*y**Integer(2)]); O
Infinite order in Function field in y defined by y^3 + x^3 + 4*x + 1
>>> O.basis()
(1, 1/x*y, 1/x^2*y^2)

```

Note that 1 does not need to be an element of the basis, as long it is in the module spanned by it:

```
sage: O = L.order_infinite_with_basis([1+1/x*y, 1/x*y, 1/x^2*y^2]); O #_
↳needs sage.rings.function_field
Infinite order in Function field in y defined by y^3 + x^3 + 4*x + 1
sage: O.basis() #_
↳needs sage.rings.function_field
(1/x*y + 1, 1/x*y, 1/x^2*y^2)
```

```
>>> from sage.all import *
>>> O = L.order_infinite_with_basis([Integer(1)+Integer(1)/x*y, Integer(1)/x*y,
↳ Integer(1)/x**Integer(2)*y**Integer(2)]); O # needs sage.rings.
↳function_field
Infinite order in Function field in y defined by y^3 + x^3 + 4*x + 1
>>> O.basis() #_
↳needs sage.rings.function_field
(1/x*y + 1, 1/x*y, 1/x^2*y^2)
```

The following error is raised when the module spanned by the basis is not closed under multiplication:

```
sage: O = L.order_infinite_with_basis([1,y, 1/x^2*y^2]); O #_
↳needs sage.rings.function_field
Traceback (most recent call last):
...
ValueError: the module generated by basis (1, y, 1/x^2*y^2) must be closed_
↳under multiplication
```

```
>>> from sage.all import *
>>> O = L.order_infinite_with_basis([Integer(1),y, Integer(1)/
↳ x**Integer(2)*y**Integer(2)]); O # needs sage.rings.
↳function_field
Traceback (most recent call last):
...
ValueError: the module generated by basis (1, y, 1/x^2*y^2) must be closed_
↳under multiplication
```

and this happens when the identity is not in the module spanned by the basis:

```
sage: O = L.order_infinite_with_basis([1/x,1/x*y, 1/x^2*y^2]) #_
↳needs sage.rings.function_field
Traceback (most recent call last):
...
ValueError: the identity element must be in the module spanned by basis (1/x,
↳ 1/x*y, 1/x^2*y^2)
```

```
>>> from sage.all import *
>>> O = L.order_infinite_with_basis([Integer(1)/x,Integer(1)/x*y, Integer(1)/
↳ x**Integer(2)*y**Integer(2)]) # needs sage.rings.function_
↳field
Traceback (most recent call last):
...
ValueError: the identity element must be in the module spanned by basis (1/x,
↳ 1/x*y, 1/x^2*y^2)
```

`order_with_basis(basis, check=True)`

Return the order with given basis over the maximal order of the base field.

INPUT:

- `basis` – list of elements of this function field
- `check` – boolean (default: `True`); if `True`, check that the basis is really linearly independent and that the module it spans is closed under multiplication, and contains the identity element.

OUTPUT: an order in the function field

EXAMPLES:

```
sage: K.<x> = FunctionField(QQ); R.<y> = K[]
sage: L.<y> = K.extension(y^3 + x^3 + 4*x + 1) #_
↪needs sage.rings.function_field
sage: O = L.order_with_basis([1, y, y^2]); O #_
↪needs sage.rings.function_field
Order in Function field in y defined by y^3 + x^3 + 4*x + 1
sage: O.basis() #_
↪needs sage.rings.function_field
(1, y, y^2)
```

```
>>> from sage.all import *
>>> K = FunctionField(QQ, names=('x',)); (x,) = K._first_ngens(1); R = K['y'];
↪ (y,) = R._first_ngens(1)
>>> L = K.extension(y**Integer(3) + x**Integer(3) + Integer(4)*x + Integer(1),
↪ names=('y',)); (y,) = L._first_ngens(1) # needs sage.rings.function_field
>>> O = L.order_with_basis([Integer(1), y, y**Integer(2)]); O #_
↪ # needs sage.rings.function_field
Order in Function field in y defined by y^3 + x^3 + 4*x + 1
>>> O.basis() #_
↪needs sage.rings.function_field
(1, y, y^2)
```

Note that 1 does not need to be an element of the basis, as long it is in the module spanned by it:

```
sage: O = L.order_with_basis([1+y, y, y^2]); O #_
↪needs sage.rings.function_field
Order in Function field in y defined by y^3 + x^3 + 4*x + 1
sage: O.basis() #_
↪needs sage.rings.function_field
(y + 1, y, y^2)
```

```
>>> from sage.all import *
>>> O = L.order_with_basis([Integer(1)+y, y, y**Integer(2)]); O #_
↪ # needs sage.rings.function_field
Order in Function field in y defined by y^3 + x^3 + 4*x + 1
>>> O.basis() #_
↪needs sage.rings.function_field
(y + 1, y, y^2)
```

The following error is raised when the module spanned by the basis is not closed under multiplication:

```
sage: O = L.order_with_basis([1, x^2 + x*y, (2/3)*y^2]); O #_
↳needs sage.rings.function_field
Traceback (most recent call last):
...
ValueError: the module generated by basis (1, x*y + x^2, 2/3*y^2) must be_
↳closed under multiplication
```

```
>>> from sage.all import *
>>> O = L.order_with_basis([Integer(1), x**Integer(2) + x*y, (Integer(2)/
↳Integer(3))*y**Integer(2)]); O # needs sage.rings.function_
↳field
Traceback (most recent call last):
...
ValueError: the module generated by basis (1, x*y + x^2, 2/3*y^2) must be_
↳closed under multiplication
```

and this happens when the identity is not in the module spanned by the basis:

```
sage: O = L.order_with_basis([x, x^2 + x*y, (2/3)*y^2]) #_
↳needs sage.rings.function_field
Traceback (most recent call last):
...
ValueError: the identity element must be in the module spanned by basis (x,
↳x*y + x^2, 2/3*y^2)
```

```
>>> from sage.all import *
>>> O = L.order_with_basis([x, x**Integer(2) + x*y, (Integer(2)/
↳Integer(3))*y**Integer(2)]); O # needs sage.rings.function_
↳field
Traceback (most recent call last):
...
ValueError: the identity element must be in the module spanned by basis (x,
↳x*y + x^2, 2/3*y^2)
```

place_set()

Return the set of all places of the function field.

EXAMPLES:

```
sage: K.<t> = FunctionField(GF(7))
sage: K.place_set()
Set of places of Rational function field in t over Finite Field of size 7

sage: K.<t> = FunctionField(QQ)
sage: K.place_set()
Set of places of Rational function field in t over Rational Field

sage: K.<x> = FunctionField(GF(2)); _.<Y> = K[]
sage: L.<y> = K.extension(Y^2 + Y + x + 1/x) #_
↳needs sage.rings.function_field
sage: L.place_set() #_
↳needs sage.rings.function_field
Set of places of Function field in y defined by y^2 + y + (x^2 + 1)/x
```



```

>>> from sage.all import *
>>> K = FunctionField(GF(Integer(7)), names=('t',)); (t,) = K._first_ngens(1)
>>> K.place_set()
Set of places of Rational function field in t over Finite Field of size 7

>>> K = FunctionField(QQ, names=('t',)); (t,) = K._first_ngens(1)
>>> K.place_set()
Set of places of Rational function field in t over Rational Field

>>> K = FunctionField(GF(Integer(2)), names=('x',)); (x,) = K._first_ngens(1);
↪ _ = K['Y']; (Y,) = _._first_ngens(1)
>>> L = K.extension(Y**Integer(2) + Y + x + Integer(1)/x, names=('y',)); (y,) ↪
↪ = L._first_ngens(1) # needs sage.rings.function_field
>>> L.place_set() # ↪
↪ needs sage.rings.function_field
Set of places of Function field in y defined by  $y^2 + y + (x^2 + 1)/x$ 

```

rational_function_field()

Return the rational function field from which this field has been created as an extension.

EXAMPLES:

```

sage: K.<x> = FunctionField(QQ)
sage: K.rational_function_field()
Rational function field in x over Rational Field

sage: R.<y> = K[]
sage: L.<y> = K.extension(y^2 - x) # ↪
↪ needs sage.rings.function_field
sage: L.rational_function_field() # ↪
↪ needs sage.rings.function_field
Rational function field in x over Rational Field

sage: R.<z> = L[] # ↪
↪ needs sage.rings.function_field
sage: M.<z> = L.extension(z^2 - y) # ↪
↪ needs sage.rings.function_field
sage: M.rational_function_field() # ↪
↪ needs sage.rings.function_field
Rational function field in x over Rational Field

```

```

>>> from sage.all import *
>>> K = FunctionField(QQ, names=('x',)); (x,) = K._first_ngens(1)
>>> K.rational_function_field()
Rational function field in x over Rational Field

>>> R = K['y']; (y,) = R._first_ngens(1)
>>> L = K.extension(y**Integer(2) - x, names=('y',)); (y,) = L._first_ngens(1)
↪ # needs sage.rings.function_field
>>> L.rational_function_field() # ↪
↪ needs sage.rings.function_field
Rational function field in x over Rational Field

```

(continues on next page)

(continued from previous page)

```

>>> R = L['z']; (z,) = R._first_ngens(1) # needs sage.rings.function_field
>>> M = L.extension(z**Integer(2) - y, names=('z',)); (z,) = M._first_ngens(1)
↪ # needs sage.rings.function_field
>>> M.rational_function_field() #_
↪ needs sage.rings.function_field
Rational function field in x over Rational Field

```

some_elements()

Return some elements in this function field.

EXAMPLES:

```

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

```

```

>>> from sage.all import *
>>> K = FunctionField(QQ, names=('x',)); (x,) = K._first_ngens(1)
>>> K.some_elements()
[1,
 x,
 2*x,
 x/(x^2 + 2*x + 1),
 1/x^2,
 x/(x^2 - 1),
 x/(x^2 + 1),
 1/2*x/(x^2 + 1),
 0,
 1/x,
 ...]

```

```

sage: R.<y> = K[]
sage: L.<y> = K.extension(y^2 - x)
↪ # needs sage.rings.function_field
sage: L.some_elements()
↪ # needs sage.rings.function_field
[1,
 y,
 1/x*y,
 ((x + 1)/(x^2 - 2*x + 1))*y - 2*x/(x^2 - 2*x + 1),
 1/x,
 (1/(x - 1))*y,
 ...]

```

(continues on next page)

(continued from previous page)

```
(1/(x + 1))*y,
(1/2/(x + 1))*y,
0,
...]
```

```
>>> from sage.all import *
>>> R = K['y']; (y,) = R._first_ngens(1)
>>> L = K.extension(y**Integer(2) - x, names=('y',)); (y,) = L._first_ngens(1)
↪ # needs sage.rings.function_field
>>> L.some_elements() #_
↪ needs sage.rings.function_field
[1,
 y,
 1/x*y,
 ((x + 1)/(x^2 - 2*x + 1))*y - 2*x/(x^2 - 2*x + 1),
 1/x,
 (1/(x - 1))*y,
 (1/(x + 1))*y,
 (1/2/(x + 1))*y,
 0,
 ...]
```

space_of_differentials()

Return the space of differentials attached to the function field.

EXAMPLES:

```
sage: K.<t> = FunctionField(QQ)
sage: K.space_of_differentials() #_
↪ needs sage.modules
Space of differentials of Rational function field in t over Rational Field

sage: K.<x> = FunctionField(GF(5)); _.<Y> = K[]
sage: L.<y> = K.extension(Y^3 - (x^3 - 1)/(x^3 - 2)) #_
↪ needs sage.rings.function_field
sage: L.space_of_differentials() #_
↪ needs sage.rings.function_field
Space of differentials of Function field in y
defined by y^3 + (4*x^3 + 1)/(x^3 + 3)
```

```
>>> from sage.all import *
>>> K = FunctionField(QQ, names=('t',)); (t,) = K._first_ngens(1)
>>> K.space_of_differentials() #_
↪ needs sage.modules
Space of differentials of Rational function field in t over Rational Field

>>> K = FunctionField(GF(Integer(5)), names=('x',)); (x,) = K._first_ngens(1);
↪ _ = K['Y']; (Y,) = _._first_ngens(1)
>>> L = K.extension(Y**Integer(3) - (x**Integer(3) - Integer(1))/
↪ (x**Integer(3) - Integer(2)), names=('y',)); (y,) = L._first_ngens(1) #_
↪ needs sage.rings.function_field
>>> L.space_of_differentials() #_
```

(continues on next page)

(continued from previous page)

```
↪needs sage.rings.function_field
Space of differentials of Function field in y
defined by  $y^3 + (4x^3 + 1)/(x^3 + 3)$ 
```

space_of_differentials_of_first_kind()

Return the space of holomorphic differentials of this function field.

EXAMPLES:

```
sage: K.<t> = FunctionField(QQ)
sage: K.space_of_holomorphic_differentials() #_
↪needs sage.libs.pari sage.modules
(Vector space of dimension 0 over Rational Field,
Linear map:
  From: Vector space of dimension 0 over Rational Field
  To:   Space of differentials of Rational function field in t over Rational
↪Field,
Section of linear map:
  From: Space of differentials of Rational function field in t over Rational
↪Field
  To:   Vector space of dimension 0 over Rational Field)

sage: K.<x> = FunctionField(GF(5)); _.<Y> = K[]
sage: L.<y> = K.extension(Y^3 - (x^3 - 1)/(x^3 - 2)) #_
↪needs sage.rings.function_field
sage: L.space_of_holomorphic_differentials() #_
↪needs sage.rings.function_field
(Vector space of dimension 4 over Finite Field of size 5,
Linear map:
  From: Vector space of dimension 4 over Finite Field of size 5
  To:   Space of differentials of Function field in y
        defined by  $y^3 + (4x^3 + 1)/(x^3 + 3)$ ,
Section of linear map:
  From: Space of differentials of Function field in y
        defined by  $y^3 + (4x^3 + 1)/(x^3 + 3)$ 
  To:   Vector space of dimension 4 over Finite Field of size 5)
```

```
>>> from sage.all import *
>>> K = FunctionField(QQ, names=('t',)); (t,) = K._first_ngens(1)
>>> K.space_of_holomorphic_differentials() #_
↪needs sage.libs.pari sage.modules
(Vector space of dimension 0 over Rational Field,
Linear map:
  From: Vector space of dimension 0 over Rational Field
  To:   Space of differentials of Rational function field in t over Rational
↪Field,
Section of linear map:
  From: Space of differentials of Rational function field in t over Rational
↪Field
  To:   Vector space of dimension 0 over Rational Field)

>>> K = FunctionField(GF(Integer(5)), names=('x',)); (x,) = K._first_ngens(1);
```

(continues on next page)

(continued from previous page)

```

↪ _ = K['Y']; (Y,) = _._first_ngens(1)
>>> L = K.extension(Y**Integer(3) - (x**Integer(3) - Integer(1))/
↪ (x**Integer(3) - Integer(2)), names=('y',)); (y,) = L._first_ngens(1) #_
↪ needs sage.rings.function_field
>>> L.space_of_holomorphic_differentials() #_
↪ needs sage.rings.function_field
(Vector space of dimension 4 over Finite Field of size 5,
Linear map:
  From: Vector space of dimension 4 over Finite Field of size 5
  To:   Space of differentials of Function field in y
        defined by  $y^3 + (4x^3 + 1)/(x^3 + 3)$ ,
Section of linear map:
  From: Space of differentials of Function field in y
        defined by  $y^3 + (4x^3 + 1)/(x^3 + 3)$ 
  To:   Vector space of dimension 4 over Finite Field of size 5)

```

space_of_holomorphic_differentials()

Return the space of holomorphic differentials of this function field.

EXAMPLES:

```

sage: K.<t> = FunctionField(QQ)
sage: K.space_of_holomorphic_differentials() #_
↪ needs sage.libs.pari sage.modules
(Vector space of dimension 0 over Rational Field,
Linear map:
  From: Vector space of dimension 0 over Rational Field
  To:   Space of differentials of Rational function field in t over Rational
↪ Field,
Section of linear map:
  From: Space of differentials of Rational function field in t over Rational
↪ Field
  To:   Vector space of dimension 0 over Rational Field)

sage: K.<x> = FunctionField(GF(5)); _.<Y> = K[]
sage: L.<y> = K.extension(Y^3 - (x^3 - 1)/(x^3 - 2)) #_
↪ needs sage.rings.function_field
sage: L.space_of_holomorphic_differentials() #_
↪ needs sage.rings.function_field
(Vector space of dimension 4 over Finite Field of size 5,
Linear map:
  From: Vector space of dimension 4 over Finite Field of size 5
  To:   Space of differentials of Function field in y
        defined by  $y^3 + (4x^3 + 1)/(x^3 + 3)$ ,
Section of linear map:
  From: Space of differentials of Function field in y
        defined by  $y^3 + (4x^3 + 1)/(x^3 + 3)$ 
  To:   Vector space of dimension 4 over Finite Field of size 5)

```

```

>>> from sage.all import *
>>> K = FunctionField(QQ, names=('t',)); (t,) = K._first_ngens(1)
>>> K.space_of_holomorphic_differentials() #_

```

(continues on next page)

(continued from previous page)

```

↪needs sage.libs.pari sage.modules
(Vector space of dimension 0 over Rational Field,
Linear map:
  From: Vector space of dimension 0 over Rational Field
  To:   Space of differentials of Rational function field in t over Rational
↪Field,
Section of linear map:
  From: Space of differentials of Rational function field in t over Rational
↪Field
  To:   Vector space of dimension 0 over Rational Field)

>>> K = FunctionField(GF(Integer(5)), names=('x',)); (x,) = K._first_ngens(1);
↪ _ = K['Y']; (Y,) = _._first_ngens(1)
>>> L = K.extension(Y**Integer(3) - (x**Integer(3) - Integer(1))/
↪(x**Integer(3) - Integer(2)), names=('y',)); (y,) = L._first_ngens(1) #
↪needs sage.rings.function_field
>>> L.space_of_holomorphic_differentials() #
↪needs sage.rings.function_field
(Vector space of dimension 4 over Finite Field of size 5,
Linear map:
  From: Vector space of dimension 4 over Finite Field of size 5
  To:   Space of differentials of Function field in y
        defined by  $y^3 + (4x^3 + 1)/(x^3 + 3)$ ,
Section of linear map:
  From: Space of differentials of Function field in y
        defined by  $y^3 + (4x^3 + 1)/(x^3 + 3)$ 
  To:   Vector space of dimension 4 over Finite Field of size 5)

```

valuation (prime)

Return the discrete valuation on this function field defined by prime.

INPUT:

- prime – a place of the function field, a valuation on a subring, or a valuation on another function field together with information for isomorphisms to and from that function field

EXAMPLES:

We create valuations that correspond to finite rational places of a function field:

```

sage: K.<x> = FunctionField(QQ)
sage: v = K.valuation(1); v #
↪needs sage.rings.function_field
(x - 1)-adic valuation
sage: v(x) #
↪needs sage.rings.function_field
0
sage: v(x - 1) #
↪needs sage.rings.function_field
1

```

```

>>> from sage.all import *
>>> K = FunctionField(QQ, names=('x',)); (x,) = K._first_ngens(1)
>>> v = K.valuation(Integer(1)); v

```

(continues on next page)

(continued from previous page)

```

↪      # needs sage.rings.function_field
(x - 1)-adic valuation
>>> v(x)                                     #_
↪needs sage.rings.function_field
0
>>> v(x - Integer(1))                       _
↪      # needs sage.rings.function_field
1

```

A place can also be specified with an irreducible polynomial:

```

sage: v = K.valuation(x - 1); v               #_
↪needs sage.rings.function_field
(x - 1)-adic valuation

```

```

>>> from sage.all import *
>>> v = K.valuation(x - Integer(1)); v        _
↪      # needs sage.rings.function_field
(x - 1)-adic valuation

```

Similarly, for a finite non-rational place:

```

sage: v = K.valuation(x^2 + 1); v             #_
↪needs sage.rings.function_field
(x^2 + 1)-adic valuation
sage: v(x^2 + 1)                             #_
↪needs sage.rings.function_field
1
sage: v(x)                                    #_
↪needs sage.rings.function_field
0

```

```

>>> from sage.all import *
>>> v = K.valuation(x**Integer(2) + Integer(1)); v _
↪      # needs sage.rings.function_field
(x^2 + 1)-adic valuation
>>> v(x**Integer(2) + Integer(1))             _
↪      # needs sage.rings.function_field
1
>>> v(x)                                     #_
↪needs sage.rings.function_field
0

```

Or for the infinite place:

```

sage: v = K.valuation(1/x); v                 #_
↪needs sage.rings.function_field
Valuation at the infinite place
sage: v(x)                                    #_
↪needs sage.rings.function_field
-1

```

```
>>> from sage.all import *
>>> v = K.valuation(Integer(1)/x); v
↪ # needs sage.rings.function_field
Valuation at the infinite place
>>> v(x)
↪ # needs sage.rings.function_field
-1
```

Instead of specifying a generator of a place, we can define a valuation on a rational function field by giving a discrete valuation on the underlying polynomial ring:

```
sage: # needs sage.rings.function_field
sage: R.<x> = QQ[]
sage: u = valuations.GaussValuation(R, valuations.TrivialValuation(QQ))
sage: w = u.augmentation(x - 1, 1)
sage: v = K.valuation(w); v
(x - 1)-adic valuation
```

```
>>> from sage.all import *
>>> # needs sage.rings.function_field
>>> R = QQ['x']; (x,) = R._first_ngens(1)
>>> u = valuations.GaussValuation(R, valuations.TrivialValuation(QQ))
>>> w = u.augmentation(x - Integer(1), Integer(1))
>>> v = K.valuation(w); v
(x - 1)-adic valuation
```

Note that this allows us to specify valuations which do not correspond to a place of the function field:

```
sage: w = valuations.GaussValuation(R, QQ.valuation(2))
↪ # needs sage.rings.function_field
sage: v = K.valuation(w); v
↪ # needs sage.rings.function_field
2-adic valuation
```

```
>>> from sage.all import *
>>> w = valuations.GaussValuation(R, QQ.valuation(Integer(2)))
↪ # needs sage.rings.function_field
>>> v = K.valuation(w); v
↪ # needs sage.rings.function_field
2-adic valuation
```

The same is possible for valuations with $v(1/x) > 0$ by passing in an extra pair of parameters, an isomorphism between this function field and an isomorphic function field. That way you can, for example, indicate that the valuation is to be understood as a valuation on $K[1/x]$, i.e., after applying the substitution $x \mapsto 1/x$ (here, the inverse map is also $x \mapsto 1/x$):

```
sage: # needs sage.rings.function_field
sage: w = valuations.GaussValuation(R, QQ.valuation(2)).augmentation(x, 1)
sage: w = K.valuation(w)
sage: v = K.valuation((w, K.hom([~K.gen()]), K.hom([~K.gen()])))
Valuation on rational function field
induced by [ Gauss valuation induced by 2-adic valuation, v(x) = 1 ]
(in Rational function field in x over Rational Field after x |--> 1/x)
```



```

>>> from sage.all import *
>>> # needs sage.rings.function_field
>>> w = valuations.GaussValuation(R, QQ.valuation(Integer(2))).augmentation(x,
↳ Integer(1))
>>> w = K.valuation(w)
>>> v = K.valuation((w, K.hom([~K.gen()]), K.hom([~K.gen()]))) ; v
Valuation on rational function field
induced by [ Gauss valuation induced by 2-adic valuation, v(x) = 1 ]
(in Rational function field in x over Rational Field after x |--> 1/x)

```

Note that classical valuations at finite places or the infinite place are always normalized such that the uniformizing element has valuation 1:

```

sage: # needs sage.rings.function_field
sage: K.<t> = FunctionField(GF(3))
sage: M.<x> = FunctionField(K)
sage: v = M.valuation(x^3 - t)
sage: v(x^3 - t)
1

```

```

>>> from sage.all import *
>>> # needs sage.rings.function_field
>>> K = FunctionField(GF(Integer(3)), names=('t',)); (t,) = K._first_ngens(1)
>>> M = FunctionField(K, names=('x',)); (x,) = M._first_ngens(1)
>>> v = M.valuation(x**Integer(3) - t)
>>> v(x**Integer(3) - t)
1

```

However, if such a valuation comes out of a base change of the ground field, this is not the case anymore. In the example below, the unique extension of v to L still has valuation 1 on $x^3 - t$ but it has valuation $1/3$ on its uniformizing element $x - w$:

```

sage: # needs sage.rings.function_field
sage: R.<w> = K[]
sage: L.<w> = K.extension(w^3 - t)
sage: N.<x> = FunctionField(L)
sage: w = v.extension(N) # missing factorization, :issue:`16572`
Traceback (most recent call last):
...
NotImplementedError
sage: w(x^3 - t) # not tested
1
sage: w(x - w) # not tested
1/3

```

```

>>> from sage.all import *
>>> # needs sage.rings.function_field
>>> R = K['w']; (w,) = R._first_ngens(1)
>>> L = K.extension(w**Integer(3) - t, names=('w',)); (w,) = L._first_ngens(1)
>>> N = FunctionField(L, names=('x',)); (x,) = N._first_ngens(1)
>>> w = v.extension(N) # missing factorization, :issue:`16572`
Traceback (most recent call last):
...

```

(continues on next page)

(continued from previous page)

```

NotImplementedError
>>> w(x**Integer(3) - t)                # not tested
1
>>> w(x - w)                            # not tested
1/3

```

There are several ways to create valuations on extensions of rational function fields:

```

sage: K.<x> = FunctionField(QQ)
sage: R.<y> = K[]
sage: L.<y> = K.extension(y^2 - x); L      #_
↪needs sage.rings.function_field
Function field in y defined by y^2 - x

```

```

>>> from sage.all import *
>>> K = FunctionField(QQ, names=('x',)); (x,) = K._first_ngens(1)
>>> R = K['y']; (y,) = R._first_ngens(1)
>>> L = K.extension(y**Integer(2) - x, names=('y',)); (y,) = L._first_
↪ngens(1); L                             # needs sage.rings.
↪function_field
Function field in y defined by y^2 - x

```

A place that has a unique extension can just be defined downstairs:

```

sage: v = L.valuation(x); v              #_
↪needs sage.rings.function_field
(x)-adic valuation

```

```

>>> from sage.all import *
>>> v = L.valuation(x); v                #_
↪needs sage.rings.function_field
(x)-adic valuation

```

`sage.rings.function_field.function_field.is_FunctionField(x)`

Return True if `x` is a function field.

EXAMPLES:

```

sage: from sage.rings.function_field.function_field import is_FunctionField
sage: is_FunctionField(QQ)
doctest:warning...
DeprecationWarning: The function is_FunctionField is deprecated; use '... in_
↪FunctionFields()' instead.
See https://github.com/sagemath/sage/issues/38289 for details.
False
sage: is_FunctionField(FunctionField(QQ, 't'))
True

```

```

>>> from sage.all import *
>>> from sage.rings.function_field.function_field import is_FunctionField
>>> is_FunctionField(QQ)
doctest:warning...

```

(continues on next page)

(continued from previous page)

```
DeprecationWarning: The function is_FunctionField is deprecated; use '... in_
↪FunctionFields()' instead.
See https://github.com/sagemath/sage/issues/38289 for details.
False
>>> is_FunctionField(FunctionField(QQ, 't'))
True
```


FUNCTION FIELDS: RATIONAL

```
class sage.rings.function_field.function_field_rational.RationalFunctionField(constant_field,
                                                                              names, category=None)
```

Bases: *FunctionField*

Rational function field in one variable, over an arbitrary base field.

INPUT:

- `constant_field` – arbitrary field
- `names` – string or tuple of length 1

EXAMPLES:

```
sage: K.<t> = FunctionField(GF(3)); K
Rational function field in t over Finite Field of size 3
sage: K.gen()
t
sage: 1/t + t^3 + 5
(t^4 + 2*t + 1)/t

sage: K.<t> = FunctionField(QQ); K
Rational function field in t over Rational Field
sage: K.gen()
t
sage: 1/t + t^3 + 5
(t^4 + 5*t + 1)/t
```

```
>>> from sage.all import *
>>> K = FunctionField(GF(Integer(3)), names=('t',)); (t,) = K._first_ngens(1); K
Rational function field in t over Finite Field of size 3
>>> K.gen()
t
>>> Integer(1)/t + t**Integer(3) + Integer(5)
(t^4 + 2*t + 1)/t

>>> K = FunctionField(QQ, names=('t',)); (t,) = K._first_ngens(1); K
Rational function field in t over Rational Field
>>> K.gen()
t
```

(continues on next page)

(continued from previous page)

```
>>> Integer(1)/t + t**Integer(3) + Integer(5)
(t^4 + 5*t + 1)/t
```

There are various ways to get at the underlying fields and rings associated to a rational function field:

```
sage: K.<t> = FunctionField(GF(7))
sage: K.base_field()
Rational function field in t over Finite Field of size 7
sage: K.field()
Fraction Field of Univariate Polynomial Ring in t over Finite Field of size 7
sage: K.constant_field()
Finite Field of size 7
sage: K.maximal_order()
Maximal order of Rational function field in t over Finite Field of size 7

sage: K.<t> = FunctionField(QQ)
sage: K.base_field()
Rational function field in t over Rational Field
sage: K.field()
Fraction Field of Univariate Polynomial Ring in t over Rational Field
sage: K.constant_field()
Rational Field
sage: K.maximal_order()
Maximal order of Rational function field in t over Rational Field
```

```
>>> from sage.all import *
>>> K = FunctionField(GF(Integer(7)), names=('t',)); (t,) = K._first_ngens(1)
>>> K.base_field()
Rational function field in t over Finite Field of size 7
>>> K.field()
Fraction Field of Univariate Polynomial Ring in t over Finite Field of size 7
>>> K.constant_field()
Finite Field of size 7
>>> K.maximal_order()
Maximal order of Rational function field in t over Finite Field of size 7

>>> K = FunctionField(QQ, names=('t',)); (t,) = K._first_ngens(1)
>>> K.base_field()
Rational function field in t over Rational Field
>>> K.field()
Fraction Field of Univariate Polynomial Ring in t over Rational Field
>>> K.constant_field()
Rational Field
>>> K.maximal_order()
Maximal order of Rational function field in t over Rational Field
```

We define a morphism:

```
sage: K.<t> = FunctionField(QQ)
sage: L = FunctionField(QQ, 'tbar') # give variable name as second input
sage: K.hom(L.gen())
Function Field morphism:
```

(continues on next page)

(continued from previous page)

```

From: Rational function field in t over Rational Field
To:   Rational function field in tbar over Rational Field
Defn: t |--> tbar

```

```

>>> from sage.all import *
>>> K = FunctionField(QQ, names=('t',)); (t,) = K._first_ngens(1)
>>> L = FunctionField(QQ, 'tbar') # give variable name as second input
>>> K.hom(L.gen())
Function Field morphism:
  From: Rational function field in t over Rational Field
  To:   Rational function field in tbar over Rational Field
  Defn: t |--> tbar

```

Here are some calculations over a number field:

```

sage: R.<x> = FunctionField(QQ)
sage: L.<y> = R[]
sage: F.<y> = R.extension(y^2 - (x^2+1)) #_
↪needs sage.rings.function_field
sage: (y/x).divisor() #_
↪needs sage.rings.function_field
- Place (x, y - 1)
- Place (x, y + 1)
+ Place (x^2 + 1, y)

sage: # needs sage.rings.number_field
sage: A.<z> = QQ[]
sage: NF.<i> = NumberField(z^2 + 1)
sage: R.<x> = FunctionField(NF)
sage: L.<y> = R[]
sage: F.<y> = R.extension(y^2 - (x^2+1)) #_
↪needs sage.rings.function_field
sage: (x/y*x.differential()).divisor() #_
↪needs sage.rings.function_field
-2*Place (1/x, 1/x*y - 1)
- 2*Place (1/x, 1/x*y + 1)
+ Place (x, y - 1)
+ Place (x, y + 1)

sage: (x/y).divisor() #_
↪needs sage.rings.function_field
- Place (x - i, y)
+ Place (x, y - 1)
+ Place (x, y + 1)
- Place (x + i, y)

```

```

>>> from sage.all import *
>>> R = FunctionField(QQ, names=('x',)); (x,) = R._first_ngens(1)
>>> L = R['y']; (y,) = L._first_ngens(1)
>>> F = R.extension(y**Integer(2) - (x**Integer(2)+Integer(1)), names=('y',)); (y,
↪) = F._first_ngens(1) # needs sage.rings.function_field
>>> (y/x).divisor() #_
↪needs sage.rings.function_field

```

(continues on next page)

(continued from previous page)

```

- Place (x, y - 1)
- Place (x, y + 1)
+ Place (x^2 + 1, y)

>>> # needs sage.rings.number_field
>>> A = QQ['z']; (z,) = A._first_ngens(1)
>>> NF = NumberField(z**Integer(2) + Integer(1), names=('i',)); (i,) = NF._first_
↳ngens(1)
>>> R = FunctionField(NF, names=('x',)); (x,) = R._first_ngens(1)
>>> L = R['y']; (y,) = L._first_ngens(1)
>>> F = R.extension(y**Integer(2) - (x**Integer(2)+Integer(1)), names=('y',)); (y,
↳) = F._first_ngens(1) # needs sage.rings.function_field
>>> (x/y*x.differential()).divisor() #↳
↳needs sage.rings.function_field
-2*Place (1/x, 1/x*y - 1)
- 2*Place (1/x, 1/x*y + 1)
+ Place (x, y - 1)
+ Place (x, y + 1)
>>> (x/y).divisor() #↳
↳needs sage.rings.function_field
- Place (x - i, y)
+ Place (x, y - 1)
+ Place (x, y + 1)
- Place (x + i, y)

```

Elementalias of `FunctionFieldElement_rational`**base_field()**

Return the base field of the rational function field, which is just the function field itself.

EXAMPLES:

```

sage: K.<t> = FunctionField(GF(7))
sage: K.base_field()
Rational function field in t over Finite Field of size 7

```

```

>>> from sage.all import *
>>> K = FunctionField(GF(Integer(7)), names=('t',)); (t,) = K._first_ngens(1)
>>> K.base_field()
Rational function field in t over Finite Field of size 7

```

change_variable_name(name)Return a field isomorphic to this field with variable `name`.**INPUT:**

- `name` – string or tuple consisting of a single string; the name of the new variable

OUTPUT:A triple F, f, t where F is a rational function field, f is an isomorphism from F to this field, and t is the inverse of f .**EXAMPLES:**


```

sage: K.<x> = FunctionField(QQ)
sage: L,f,t = K.change_variable_name('y')
sage: L,f,t
(Rational function field in y over Rational Field,
Function Field morphism:
  From: Rational function field in y over Rational Field
  To:   Rational function field in x over Rational Field
  Defn: y |--> x,
Function Field morphism:
  From: Rational function field in x over Rational Field
  To:   Rational function field in y over Rational Field
  Defn: x |--> y)
sage: L.change_variable_name('x')[0] is K
True

```

```

>>> from sage.all import *
>>> K = FunctionField(QQ, names=('x',)); (x,) = K._first_ngens(1)
>>> L,f,t = K.change_variable_name('y')
>>> L,f,t
(Rational function field in y over Rational Field,
Function Field morphism:
  From: Rational function field in y over Rational Field
  To:   Rational function field in x over Rational Field
  Defn: y |--> x,
Function Field morphism:
  From: Rational function field in x over Rational Field
  To:   Rational function field in y over Rational Field
  Defn: x |--> y)
>>> L.change_variable_name('x')[Integer(0)] is K
True

```

constant_base_field()

Return the field of which the rational function field is a transcendental extension.

EXAMPLES:

```

sage: K.<t> = FunctionField(QQ)
sage: K.constant_base_field()
Rational Field

```

```

>>> from sage.all import *
>>> K = FunctionField(QQ, names=('t',)); (t,) = K._first_ngens(1)
>>> K.constant_base_field()
Rational Field

```

constant_field()

Return the field of which the rational function field is a transcendental extension.

EXAMPLES:

```

sage: K.<t> = FunctionField(QQ)
sage: K.constant_base_field()
Rational Field

```

```
>>> from sage.all import *
>>> K = FunctionField(QQ, names=('t',)); (t,) = K._first_ngens(1)
>>> K.constant_base_field()
Rational Field
```

degree (*base=None*)

Return the degree over the base field of the rational function field. Since the base field is the rational function field itself, the degree is 1.

INPUT:

- *base* – the base field of the vector space; must be the function field itself (the default)

EXAMPLES:

```
sage: K.<t> = FunctionField(QQ)
sage: K.degree()
1
```

```
>>> from sage.all import *
>>> K = FunctionField(QQ, names=('t',)); (t,) = K._first_ngens(1)
>>> K.degree()
1
```

different ()

Return the different of the rational function field.

For a rational function field, the different is simply the zero divisor.

EXAMPLES:

```
sage: K.<t> = FunctionField(QQ)
sage: K.different()
↪needs sage.modules
0
```

#

```
>>> from sage.all import *
>>> K = FunctionField(QQ, names=('t',)); (t,) = K._first_ngens(1)
>>> K.different()
↪needs sage.modules
0
```

#

equation_order ()

Return the maximal order of the function field.

Since this is a rational function field it is of the form $K(t)$, and the maximal order is by definition $K[t]$, where K is the constant field.

EXAMPLES:

```
sage: K.<t> = FunctionField(QQ)
sage: K.maximal_order()
Maximal order of Rational function field in t over Rational Field
sage: K.equation_order()
Maximal order of Rational function field in t over Rational Field
```

```
>>> from sage.all import *
>>> K = FunctionField(QQ, names=('t',)); (t,) = K._first_ngens(1)
>>> K.maximal_order()
Maximal order of Rational function field in t over Rational Field
>>> K.equation_order()
Maximal order of Rational function field in t over Rational Field
```

equation_order_infinite()

Return the maximal infinite order of the function field.

By definition, this is the valuation ring of the degree valuation of the rational function field.

EXAMPLES:

```
sage: K.<t> = FunctionField(QQ)
sage: K.maximal_order_infinite()
Maximal infinite order of Rational function field in t over Rational Field
sage: K.equation_order_infinite()
Maximal infinite order of Rational function field in t over Rational Field
```

```
>>> from sage.all import *
>>> K = FunctionField(QQ, names=('t',)); (t,) = K._first_ngens(1)
>>> K.maximal_order_infinite()
Maximal infinite order of Rational function field in t over Rational Field
>>> K.equation_order_infinite()
Maximal infinite order of Rational function field in t over Rational Field
```

extension(f, names=None)

Create an extension $L = K[y]/(f(y))$ of the rational function field.

INPUT:

- f – univariate polynomial over self
- $names$ – string or length-1 tuple

OUTPUT: a function field

EXAMPLES:

```
sage: K.<x> = FunctionField(QQ); R.<y> = K[]
sage: K.extension(y^5 - x^3 - 3*x + x*y) #_
↪ needs sage.rings.function_field
Function field in y defined by y^5 + x*y - x^3 - 3*x
```

```
>>> from sage.all import *
>>> K = FunctionField(QQ, names=('x',)); (x,) = K._first_ngens(1); R = K['y'];
↪ (y,) = R._first_ngens(1)
>>> K.extension(y**Integer(5) - x**Integer(3) - Integer(3)*x + x*y) #_
↪ # needs sage.rings.function_field
Function field in y defined by y^5 + x*y - x^3 - 3*x
```

A nonintegral defining polynomial:

```
sage: K.<t> = FunctionField(QQ); R.<y> = K[]
sage: K.extension(y^3 + (1/t)*y + t^3/(t+1)) #_
```

(continues on next page)

(continued from previous page)

```
↪needs sage.rings.function_field
Function field in y defined by  $y^3 + 1/t*y + t^3/(t + 1)$ 
```

```
>>> from sage.all import *
>>> K = FunctionField(QQ, names=('t',)); (t,) = K._first_ngens(1); R = K['y'];
↪ (y,) = R._first_ngens(1)
>>> K.extension(y**Integer(3) + (Integer(1)/t)*y + t**Integer(3)/
↪ (t+Integer(1))) # needs sage.rings.function_
↪field
Function field in y defined by  $y^3 + 1/t*y + t^3/(t + 1)$ 
```

The defining polynomial need not be monic or integral:

```
sage: K.extension(t*y^3 + (1/t)*y + t^3/(t+1)) #_
↪needs sage.rings.function_field
Function field in y defined by  $t*y^3 + 1/t*y + t^3/(t + 1)$ 
```

```
>>> from sage.all import *
>>> K.extension(t*y**Integer(3) + (Integer(1)/t)*y + t**Integer(3)/
↪ (t+Integer(1))) # needs sage.rings.function_
↪field
Function field in y defined by  $t*y^3 + 1/t*y + t^3/(t + 1)$ 
```

field()

Return the underlying field, forgetting the function field structure.

EXAMPLES:

```
sage: K.<t> = FunctionField(GF(7))
sage: K.field()
Fraction Field of Univariate Polynomial Ring in t over Finite Field of size 7
```

```
>>> from sage.all import *
>>> K = FunctionField(GF(Integer(7)), names=('t',)); (t,) = K._first_ngens(1)
>>> K.field()
Fraction Field of Univariate Polynomial Ring in t over Finite Field of size 7
```

See also

```
sage.rings.fraction_field.FractionField_1poly_field.function_field()
```

free_module (*base=None, basis=None, map=True*)

Return a vector space V and isomorphisms from the field to V and from V to the field.

This function allows us to identify the elements of this field with elements of a one-dimensional vector space over the field itself. This method exists so that all function fields (rational or not) have the same interface.

INPUT:

- *base* – the base field of the vector space; must be the function field itself (the default)
- *basis* – (ignored) a basis for the vector space
- *map* – (default: `True`) whether to return maps to and from the vector space

OUTPUT:

- a vector space V over base field
- an isomorphism from V to the field
- the inverse isomorphism from the field to V

EXAMPLES:

```
sage: K.<x> = FunctionField(QQ)
sage: K.free_module()
↪                                     # needs sage.modules
(Vector space of dimension 1 over Rational function field in x over Rational
↪Field,
Isomorphism:
  From: Vector space of dimension 1 over Rational function field in x over
↪Rational Field
  To:   Rational function field in x over Rational Field,
Isomorphism:
  From: Rational function field in x over Rational Field
  To:   Vector space of dimension 1 over Rational function field in x over
↪Rational Field)
```

```
>>> from sage.all import *
>>> K = FunctionField(QQ, names=('x',)); (x,) = K._first_ngens(1)
>>> K.free_module()
↪                                     # needs sage.modules
(Vector space of dimension 1 over Rational function field in x over Rational
↪Field,
Isomorphism:
  From: Vector space of dimension 1 over Rational function field in x over
↪Rational Field
  To:   Rational function field in x over Rational Field,
Isomorphism:
  From: Rational function field in x over Rational Field
  To:   Vector space of dimension 1 over Rational function field in x over
↪Rational Field)
```

gen ($n=0$)

Return the n -th generator of the function field. If n is not 0, then an `:class:`IndexError`` is raised.

EXAMPLES:

```
sage: K.<t> = FunctionField(QQ); K.gen()
t
sage: K.gen().parent()
Rational function field in t over Rational Field
sage: K.gen(1)
Traceback (most recent call last):
...
IndexError: Only one generator.
```

```
>>> from sage.all import *
>>> K = FunctionField(QQ, names=('t',)); (t,) = K._first_ngens(1); K.gen()
```

(continues on next page)

(continued from previous page)

```
t
>>> K.gen().parent()
Rational function field in t over Rational Field
>>> K.gen(Integer(1))
Traceback (most recent call last):
...
IndexError: Only one generator.
```

genus()

Return the genus of the function field, namely 0.

EXAMPLES:

```
sage: K.<x> = FunctionField(QQ)
sage: K.genus()
0
```

```
>>> from sage.all import *
>>> K = FunctionField(QQ, names=('x',)); (x,) = K._first_ngens(1)
>>> K.genus()
0
```

hom(im_gens, base_morphism=None)

Create a homomorphism from `self` to another ring.

INPUT:

- `im_gens` – exactly one element of some ring. It must be invertible and transcendental over the image of `base_morphism`; this is not checked.
- `base_morphism` – a homomorphism from the base field into the other ring; if `None`, try to use a coercion map

OUTPUT: a map between function fields

EXAMPLES:

We make a map from a rational function field to itself:

```
sage: K.<x> = FunctionField(GF(7))
sage: K.hom((x^4 + 2)/x)
Function Field endomorphism of Rational function field in x over Finite Field_
↳ of size 7
Defn: x |--> (x^4 + 2)/x
```

```
>>> from sage.all import *
>>> K = FunctionField(GF(Integer(7)), names=('x',)); (x,) = K._first_ngens(1)
>>> K.hom((x**Integer(4) + Integer(2))/x)
Function Field endomorphism of Rational function field in x over Finite Field_
↳ of size 7
Defn: x |--> (x^4 + 2)/x
```

We construct a map from a rational function field into a non-rational extension field:

```

sage: # needs sage.rings.function_field
sage: K.<x> = FunctionField(GF(7)); R.<y> = K[]
sage: L.<y> = K.extension(y^3 + 6*x^3 + x)
sage: f = K.hom(y^2 + y + 2); f
Function Field morphism:
  From: Rational function field in x over Finite Field of size 7
  To:   Function field in y defined by y^3 + 6*x^3 + x
  Defn: x |--> y^2 + y + 2
sage: f(x)
y^2 + y + 2
sage: f(x^2)
5*y^2 + (x^3 + 6*x + 4)*y + 2*x^3 + 5*x + 4

```

```

>>> from sage.all import *
>>> # needs sage.rings.function_field
>>> K = FunctionField(GF(Integer(7)), names=('x',)); (x,) = K._first_ngens(1);
↪ R = K['y']; (y,) = R._first_ngens(1)
>>> L = K.extension(y**Integer(3) + Integer(6)*x**Integer(3) + x, names=('y',
↪ )); (y,) = L._first_ngens(1)
>>> f = K.hom(y**Integer(2) + y + Integer(2)); f
Function Field morphism:
  From: Rational function field in x over Finite Field of size 7
  To:   Function field in y defined by y^3 + 6*x^3 + x
  Defn: x |--> y^2 + y + 2
>>> f(x)
y^2 + y + 2
>>> f(x**Integer(2))
5*y^2 + (x^3 + 6*x + 4)*y + 2*x^3 + 5*x + 4

```

maximal_order()

Return the maximal order of the function field.

Since this is a rational function field it is of the form $K(t)$, and the maximal order is by definition $K[t]$, where K is the constant field.

EXAMPLES:

```

sage: K.<t> = FunctionField(QQ)
sage: K.maximal_order()
Maximal order of Rational function field in t over Rational Field
sage: K.equation_order()
Maximal order of Rational function field in t over Rational Field

```

```

>>> from sage.all import *
>>> K = FunctionField(QQ, names=('t',)); (t,) = K._first_ngens(1)
>>> K.maximal_order()
Maximal order of Rational function field in t over Rational Field
>>> K.equation_order()
Maximal order of Rational function field in t over Rational Field

```

maximal_order_infinite()

Return the maximal infinite order of the function field.

By definition, this is the valuation ring of the degree valuation of the rational function field.

EXAMPLES:

```
sage: K.<t> = FunctionField(QQ)
sage: K.maximal_order_infinite()
Maximal infinite order of Rational function field in t over Rational Field
sage: K.equation_order_infinite()
Maximal infinite order of Rational function field in t over Rational Field
```

```
>>> from sage.all import *
>>> K = FunctionField(QQ, names=('t',)); (t,) = K._first_ngens(1)
>>> K.maximal_order_infinite()
Maximal infinite order of Rational function field in t over Rational Field
>>> K.equation_order_infinite()
Maximal infinite order of Rational function field in t over Rational Field
```

ngens()

Return the number of generators, which is 1.

EXAMPLES:

```
sage: K.<t> = FunctionField(QQ)
sage: K.ngens()
1
```

```
>>> from sage.all import *
>>> K = FunctionField(QQ, names=('t',)); (t,) = K._first_ngens(1)
>>> K.ngens()
1
```

polynomial_ring(var='x')

Return a polynomial ring in one variable over the rational function field.

INPUT:

- var – string; name of the variable

EXAMPLES:

```
sage: K.<x> = FunctionField(QQ)
sage: K.polynomial_ring()
Univariate Polynomial Ring in x over Rational function field in x over
↳ Rational Field
sage: K.polynomial_ring('T')
Univariate Polynomial Ring in T over Rational function field in x over
↳ Rational Field
```

```
>>> from sage.all import *
>>> K = FunctionField(QQ, names=('x',)); (x,) = K._first_ngens(1)
>>> K.polynomial_ring()
Univariate Polynomial Ring in x over Rational function field in x over
↳ Rational Field
>>> K.polynomial_ring('T')
Univariate Polynomial Ring in T over Rational function field in x over
↳ Rational Field
```


random_element (*args, **kwargs)

Create a random element of the rational function field.

Parameters are passed to the random_element method of the underlying fraction field.

EXAMPLES:

```
sage: FunctionField(QQ, 'alpha').random_element() # random
(-1/2*alpha^2 - 4)/(-12*alpha^2 + 1/2*alpha - 1/95)
```

```
>>> from sage.all import *
>>> FunctionField(QQ, 'alpha').random_element() # random
(-1/2*alpha^2 - 4)/(-12*alpha^2 + 1/2*alpha - 1/95)
```

residue_field (place, name=None)

Return the residue field of the place along with the maps from and to it.

INPUT:

- place – place of the function field
- name – string; name of the generator of the residue field

EXAMPLES:

```
sage: F.<x> = FunctionField(GF(5))
sage: p = F.places_finite(2)[0] #_
↪needs sage.libs.pari
sage: R, fr_R, to_R = F.residue_field(p) #_
↪needs sage.libs.pari sage.rings.function_field
sage: R #_
↪needs sage.libs.pari sage.rings.function_field
Finite Field in z2 of size 5^2
sage: to_R(x) in R #_
↪needs sage.libs.pari sage.rings.function_field
True
```

```
>>> from sage.all import *
>>> F = FunctionField(GF(Integer(5)), names=('x',)); (x,) = F._first_ngens(1)
>>> p = F.places_finite(Integer(2))[Integer(0)] _
↪ # needs sage.libs.pari
>>> R, fr_R, to_R = F.residue_field(p) #_
↪needs sage.libs.pari sage.rings.function_field
>>> R #_
↪needs sage.libs.pari sage.rings.function_field
Finite Field in z2 of size 5^2
>>> to_R(x) in R #_
↪needs sage.libs.pari sage.rings.function_field
True
```

```
class sage.rings.function_field.function_field_rational.RationalFunctionField_char_zero (con-
stant_field,
names,
cat-
e-
gory=None)
```

Bases: *RationalFunctionField*

Rational function fields of characteristic zero.

higher_derivation()

Return the higher derivation for the function field.

This is also called the Hasse-Schmidt derivation.

EXAMPLES:

```
sage: F.<x> = FunctionField(QQ)
sage: d = F.higher_derivation()                                     #_
↪needs sage.libs.singular sage.modules
sage: [d(x^5,i) for i in range(10)]                               #_
↪needs sage.libs.singular sage.modules
[x^5, 5*x^4, 10*x^3, 10*x^2, 5*x, 1, 0, 0, 0, 0]
sage: [d(x^9,i) for i in range(10)]                               #_
↪needs sage.libs.singular sage.modules
[x^9, 9*x^8, 36*x^7, 84*x^6, 126*x^5, 126*x^4, 84*x^3, 36*x^2, 9*x, 1]
```

```
>>> from sage.all import *
>>> F = FunctionField(QQ, names=('x',)); (x,) = F._first_ngens(1)
>>> d = F.higher_derivation()                                     #_
↪needs sage.libs.singular sage.modules
>>> [d(x**Integer(5),i) for i in range(Integer(10))]              _
↪                               # needs sage.libs.singular sage.modules
[x^5, 5*x^4, 10*x^3, 10*x^2, 5*x, 1, 0, 0, 0, 0]
>>> [d(x**Integer(9),i) for i in range(Integer(10))]              _
↪                               # needs sage.libs.singular sage.modules
[x^9, 9*x^8, 36*x^7, 84*x^6, 126*x^5, 126*x^4, 84*x^3, 36*x^2, 9*x, 1]
```

```
class sage.rings.function_field.function_field_rational.RationalFunctionField_global(constant_field,
names,
cat-
e-
gory=None)
```

Bases: *RationalFunctionField*

Rational function field over finite fields.

get_place(degree)

Return a place of degree.

INPUT:

- degree – positive integer

EXAMPLES:

```
sage: F.<a> = GF(2)
sage: K.<x> = FunctionField(F)
sage: K.get_place(1)                                             #_
↪needs sage.libs.pari
Place (x)
```

(continues on next page)

(continued from previous page)

```

sage: K.get_place(2)                                     #_
↪needs sage.libs.pari
Place (x^2 + x + 1)
sage: K.get_place(3)                                     #_
↪needs sage.libs.pari
Place (x^3 + x + 1)
sage: K.get_place(4)                                     #_
↪needs sage.libs.pari
Place (x^4 + x + 1)
sage: K.get_place(5)                                     #_
↪needs sage.libs.pari
Place (x^5 + x^2 + 1)

```

```

>>> from sage.all import *
>>> F = GF(Integer(2), names=('a',)); (a,) = F._first_ngens(1)
>>> K = FunctionField(F, names=('x',)); (x,) = K._first_ngens(1)
>>> K.get_place(Integer(1))                               _
↪      # needs sage.libs.pari
Place (x)
>>> K.get_place(Integer(2))                               _
↪      # needs sage.libs.pari
Place (x^2 + x + 1)
>>> K.get_place(Integer(3))                               _
↪      # needs sage.libs.pari
Place (x^3 + x + 1)
>>> K.get_place(Integer(4))                               _
↪      # needs sage.libs.pari
Place (x^4 + x + 1)
>>> K.get_place(Integer(5))                               _
↪      # needs sage.libs.pari
Place (x^5 + x^2 + 1)

```

higher_derivation()

Return the higher derivation for the function field.

This is also called the Hasse-Schmidt derivation.

EXAMPLES:

```

sage: F.<x> = FunctionField(GF(5))
sage: d = F.higher_derivation()                           #_
↪needs sage.rings.function_field
sage: [d(x^5,i) for i in range(10)]                       #_
↪needs sage.rings.function_field
[x^5, 0, 0, 0, 0, 0, 1, 0, 0, 0]
sage: [d(x^7,i) for i in range(10)]                       #_
↪needs sage.rings.function_field
[x^7, 2*x^6, x^5, 0, 0, x^2, 2*x, 1, 0, 0]

```

```

>>> from sage.all import *
>>> F = FunctionField(GF(Integer(5)), names=('x',)); (x,) = F._first_ngens(1)
>>> d = F.higher_derivation()                               #_

```

(continues on next page)

(continued from previous page)

```

↪needs sage.rings.function_field
>>> [d(x**Integer(5),i) for i in range(Integer(10))]
↪ # needs sage.rings.function_field
[x^5, 0, 0, 0, 0, 1, 0, 0, 0, 0]
>>> [d(x**Integer(7),i) for i in range(Integer(10))]
↪ # needs sage.rings.function_field
[x^7, 2*x^6, x^5, 0, 0, x^2, 2*x, 1, 0, 0]

```

place_infinite()

Return the unique place at infinity.

EXAMPLES:

```

sage: F.<x> = FunctionField(GF(5))
sage: F.place_infinite()
Place (1/x)

```

```

>>> from sage.all import *
>>> F = FunctionField(GF(Integer(5)), names=('x',)); (x,) = F._first_ngens(1)
>>> F.place_infinite()
Place (1/x)

```

places (degree=1)

Return all places of the degree.

INPUT:

- degree – (default: 1) a positive integer

EXAMPLES:

```

sage: F.<x> = FunctionField(GF(5))
sage: F.places()
↪needs sage.libs.pari #_
[Place (1/x),
 Place (x),
 Place (x + 1),
 Place (x + 2),
 Place (x + 3),
 Place (x + 4)]

```

```

>>> from sage.all import *
>>> F = FunctionField(GF(Integer(5)), names=('x',)); (x,) = F._first_ngens(1)
>>> F.places()
↪needs sage.libs.pari #_
[Place (1/x),
 Place (x),
 Place (x + 1),
 Place (x + 2),
 Place (x + 3),
 Place (x + 4)]

```

places_finite (degree=1)

Return the finite places of the degree.

INPUT:

- degree – (default: 1) a positive integer

EXAMPLES:

```
sage: F.<x> = FunctionField(GF(5))
sage: F.places_finite()
↳needs sage.libs.pari
[Place (x), Place (x + 1), Place (x + 2), Place (x + 3), Place (x + 4)]
```

```
>>> from sage.all import *
>>> F = FunctionField(GF(Integer(5)), names=('x',)); (x,) = F._first_ngens(1)
>>> F.places_finite()
↳needs sage.libs.pari
[Place (x), Place (x + 1), Place (x + 2), Place (x + 3), Place (x + 4)]
```


FUNCTION FIELDS: EXTENSION

```
class sage.rings.function_field.function_field_polymod.FunctionField_char_zero(polynomial,
                                                                              names,
                                                                              category=None)
```

Bases: *FunctionField_simple*

Function fields of characteristic zero.

EXAMPLES:

```
sage: K.<x> = FunctionField(QQ); _.<Y> = K[]
sage: L.<y> = K.extension(Y^3 - (x^3 - 1)/(x^3 - 2))
sage: L
Function field in y defined by y^3 + (-x^3 + 1)/(x^3 - 2)
sage: L.characteristic()
0
```

```
>>> from sage.all import *
>>> K = FunctionField(QQ, names=('x',)); (x,) = K._first_ngens(1); _ = K['Y']; (Y,
↪) = _.first_ngens(1)
>>> L = K.extension(Y**Integer(3) - (x**Integer(3) - Integer(1))/(x**Integer(3) -
↪Integer(2)), names=('y',)); (y,) = L._first_ngens(1)
>>> L
Function field in y defined by y^3 + (-x^3 + 1)/(x^3 - 2)
>>> L.characteristic()
0
```

higher_derivation()

Return the higher derivation (also called the Hasse-Schmidt derivation) for the function field.

The higher derivation of the function field is uniquely determined with respect to the separating element x of the base rational function field $k(x)$.

EXAMPLES:

```
sage: K.<x> = FunctionField(QQ); _.<Y> = K[]
sage: L.<y> = K.extension(Y^3 - (x^3 - 1)/(x^3 - 2))
sage: L.higher_derivation()
↪needs sage.modules
Higher derivation map:
```

(continues on next page)

(continued from previous page)

From: Function field in y defined by $y^3 + (-x^3 + 1)/(x^3 - 2)$
 To: Function field in y defined by $y^3 + (-x^3 + 1)/(x^3 - 2)$

```
>>> from sage.all import *
>>> K = FunctionField(QQ, names=('x',)); (x,) = K._first_ngens(1); _ = K['Y'];
↳ (Y,) = _._first_ngens(1)
>>> L = K.extension(Y**Integer(3) - (x**Integer(3) - Integer(1))/
↳ (x**Integer(3) - Integer(2)), names=('y',)); (y,) = L._first_ngens(1)
>>> L.higher_derivation() #_
↳ needs sage.modules
Higher derivation map:
  From: Function field in  $y$  defined by  $y^3 + (-x^3 + 1)/(x^3 - 2)$ 
  To: Function field in  $y$  defined by  $y^3 + (-x^3 + 1)/(x^3 - 2)$ 
```

class sage.rings.function_field.function_field_polymod.**FunctionField_char_zero_integral** (*poly-
no-
mial,
names,
cat-
e-
gory=None*)

Bases: *FunctionField_char_zero, FunctionField_integral*

Function fields of characteristic zero, defined by an irreducible and separable polynomial, integral over the maximal order of the base rational function field with a finite constant field.

class sage.rings.function_field.function_field_polymod.**FunctionField_global** (*polynomial,
names*)

Bases: *FunctionField_simple*

Global function fields.

INPUT:

- polynomial – monic irreducible and separable polynomial
- names – name of the generator of the function field

EXAMPLES:

```
sage: K.<x> = FunctionField(GF(5)); _.<Y> = K[] #_
↳ needs sage.rings.finite_rings
sage: L.<y> = K.extension(Y^3 - (x^3 - 1)/(x^3 - 2)) #_
↳ needs sage.rings.finite_rings
sage: L #_
↳ needs sage.rings.finite_rings
Function field in  $y$  defined by  $y^3 + (4x^3 + 1)/(x^3 + 3)$ 
```

```
>>> from sage.all import *
>>> K = FunctionField(GF(Integer(5)), names=('x',)); (x,) = K._first_ngens(1); _ =
↳ K['Y']; (Y,) = _._first_ngens(1) # needs sage.rings.finite_rings
>>> L = K.extension(Y**Integer(3) - (x**Integer(3) - Integer(1))/(x**Integer(3) -
↳ Integer(2)), names=('y',)); (y,) = L._first_ngens(1) # needs sage.rings.finite_
↳ rings
```

(continues on next page)

(continued from previous page)

```
>>> L
↳needs sage.rings.finite_rings
Function field in y defined by  $y^3 + (4x^3 + 1)/(x^3 + 3)$ 
```

The defining equation needs not be monic:

```
sage: K.<x> = FunctionField(GF(4)); _.<Y> = K[]
↳needs sage.rings.finite_rings
sage: L.<y> = K.extension((1 - x)*Y^7 - x^3)
↳needs sage.rings.finite_rings
sage: L.gaps() # long time (6s)
↳needs sage.rings.finite_rings
[1, 2, 3]
```

```
>>> from sage.all import *
>>> K = FunctionField(GF(Integer(4)), names=('x',)); (x,) = K._first_ngens(1); _
↳= K['Y']; (Y,) = _._first_ngens(1) # needs sage.rings.finite_rings
>>> L = K.extension((Integer(1) - x)*Y**Integer(7) - x**Integer(3), names=('y',));
↳ (y,) = L._first_ngens(1) # needs sage.rings.finite_rings
>>> L.gaps() # long time (6s)
↳needs sage.rings.finite_rings
[1, 2, 3]
```

or may define a trivial extension:

```
sage: K.<x> = FunctionField(GF(5)); _.<Y> = K[]
↳needs sage.rings.finite_rings
sage: L.<y> = K.extension(Y-1)
↳needs sage.rings.finite_rings
sage: L.genus()
↳needs sage.rings.finite_rings
0
```

```
>>> from sage.all import *
>>> K = FunctionField(GF(Integer(5)), names=('x',)); (x,) = K._first_ngens(1); _
↳= K['Y']; (Y,) = _._first_ngens(1) # needs sage.rings.finite_rings
>>> L = K.extension(Y-Integer(1), names=('y',)); (y,) = L._first_ngens(1) # needs
↳sage.rings.finite_rings
>>> L.genus()
↳needs sage.rings.finite_rings
0
```

L_polynomial (name='t')

Return the L-polynomial of the function field.

INPUT:

- name – (default: t) name of the variable of the polynomial

EXAMPLES:

```
sage: K.<x> = FunctionField(GF(2)); _.<Y> = K[]
↳needs sage.rings.finite_rings
```

(continues on next page)

(continued from previous page)

```
sage: F.<y> = K.extension(Y^2 + Y + x + 1/x) #_
↳needs sage.rings.finite_rings
sage: F.L_polynomial() #_
↳needs sage.rings.finite_rings
2*t^2 + t + 1
```

```
>>> from sage.all import *
>>> K = FunctionField(GF(Integer(2)), names=('x',)); (x,) = K._first_ngens(1);
↳ _ = K['Y']; (Y,) = _._first_ngens(1) # needs sage.rings.finite_rings
>>> F = K.extension(Y**Integer(2) + Y + x + Integer(1)/x, names=('y',)); (y,) =
↳ F._first_ngens(1) # needs sage.rings.finite_rings
>>> F.L_polynomial() #_
↳needs sage.rings.finite_rings
2*t^2 + t + 1
```

gaps()

Return the gaps of the function field.

These are the gaps at the ordinary places, that is, places which are not Weierstrass places.

EXAMPLES:

```
sage: K.<x> = FunctionField(GF(2)); _.<Y> = K[] #_
↳needs sage.rings.finite_rings
sage: L.<y> = K.extension(Y^3 + x^3 * Y + x) #_
↳needs sage.rings.finite_rings
sage: L.gaps() #_
↳needs sage.modules sage.rings.finite_rings
[1, 2, 3]
```

```
>>> from sage.all import *
>>> K = FunctionField(GF(Integer(2)), names=('x',)); (x,) = K._first_ngens(1);
↳ _ = K['Y']; (Y,) = _._first_ngens(1) # needs sage.rings.finite_rings
>>> L = K.extension(Y**Integer(3) + x**Integer(3) * Y + x, names=('y',)); (y,
↳ ) = L._first_ngens(1) # needs sage.rings.finite_rings
>>> L.gaps() #_
↳needs sage.modules sage.rings.finite_rings
[1, 2, 3]
```

get_place (degree)

Return a place of degree.

INPUT:

- degree – positive integer

OUTPUT: a place of degree if any exists; otherwise None

EXAMPLES:

```
sage: # needs sage.rings.finite_rings
sage: F.<a> = GF(2)
sage: K.<x> = FunctionField(F)
sage: R.<Y> = PolynomialRing(K)
```

(continues on next page)

(continued from previous page)

```

sage: L.<y> = K.extension(Y^4 + Y - x^5)
sage: L.get_place(1)
Place (x, y)
sage: L.get_place(2)
Place (x, y^2 + y + 1)
sage: L.get_place(3)
Place (x^3 + x^2 + 1, y + x^2 + x)
sage: L.get_place(4)
Place (x + 1, x^5 + 1)
sage: L.get_place(5)
Place (x^5 + x^3 + x^2 + x + 1, y + x^4 + 1)
sage: L.get_place(6)
Place (x^3 + x^2 + 1, y^2 + y + x^2)
sage: L.get_place(7)
Place (x^7 + x + 1, y + x^6 + x^5 + x^4 + x^3 + x)
sage: L.get_place(8)

```

```

>>> from sage.all import *
>>> # needs sage.rings.finite_rings
>>> F = GF(Integer(2), names=('a',)); (a,) = F._first_ngens(1)
>>> K = FunctionField(F, names=('x',)); (x,) = K._first_ngens(1)
>>> R = PolynomialRing(K, names=('Y',)); (Y,) = R._first_ngens(1)
>>> L = K.extension(Y**Integer(4) + Y - x**Integer(5), names=('y',)); (y,) = L._first_ngens(1)
>>> L.get_place(Integer(1))
Place (x, y)
>>> L.get_place(Integer(2))
Place (x, y^2 + y + 1)
>>> L.get_place(Integer(3))
Place (x^3 + x^2 + 1, y + x^2 + x)
>>> L.get_place(Integer(4))
Place (x + 1, x^5 + 1)
>>> L.get_place(Integer(5))
Place (x^5 + x^3 + x^2 + x + 1, y + x^4 + 1)
>>> L.get_place(Integer(6))
Place (x^3 + x^2 + 1, y^2 + y + x^2)
>>> L.get_place(Integer(7))
Place (x^7 + x + 1, y + x^6 + x^5 + x^4 + x^3 + x)
>>> L.get_place(Integer(8))

```

higher_derivation()

Return the higher derivation (also called the Hasse-Schmidt derivation) for the function field.

The higher derivation of the function field is uniquely determined with respect to the separating element x of the base rational function field $k(x)$.

EXAMPLES:

```

sage: K.<x> = FunctionField(GF(5)); L.<Y> = K[] #_
<needs sage.rings.finite_rings>
sage: L.<y> = K.extension(Y^3 - (x^3 - 1)/(x^3 - 2)) #_
<needs sage.rings.finite_rings>
sage: L.higher_derivation() #_

```

(continues on next page)

(continued from previous page)

```
↪needs sage.modules.sage.rings.finite_rings
```

Higher derivation map:

From: Function field in y defined by $y^3 + (4x^3 + 1)/(x^3 + 3)$

To: Function field in y defined by $y^3 + (4x^3 + 1)/(x^3 + 3)$

```
>>> from sage.all import *
>>> K = FunctionField(GF(Integer(5)), names=('x',)); (x,) = K._first_ngens(1);
↪ _ = K['Y']; (Y,) = _._first_ngens(1) # needs sage.rings.finite_rings
>>> L = K.extension(Y**Integer(3) - (x**Integer(3) - Integer(1))/
↪ (x**Integer(3) - Integer(2)), names=('y',)); (y,) = L._first_ngens(1) #_
↪needs sage.rings.finite_rings
>>> L.higher_derivation() #_
↪needs sage.modules.sage.rings.finite_rings
Higher derivation map:
From: Function field in  $y$  defined by  $y^3 + (4x^3 + 1)/(x^3 + 3)$ 
To: Function field in  $y$  defined by  $y^3 + (4x^3 + 1)/(x^3 + 3)$ 
```

maximal_order()

Return the maximal order of the function field.

EXAMPLES:

```
sage: # needs sage.rings.finite_rings
sage: K.<x> = FunctionField(GF(2))
sage: R.<t> = PolynomialRing(K)
sage: F.<y> = K.extension(t^4 + x^12*t^2 + x^18*t + x^21 + x^18)
sage: O = F.maximal_order()
sage: O.basis()
(1, 1/x^4*y, 1/x^11*y^2 + 1/x^2, 1/x^15*y^3 + 1/x^6*y)
```

```
>>> from sage.all import *
>>> # needs sage.rings.finite_rings
>>> K = FunctionField(GF(Integer(2)), names=('x',)); (x,) = K._first_ngens(1)
>>> R = PolynomialRing(K, names=('t',)); (t,) = R._first_ngens(1)
>>> F = K.extension(t**Integer(4) + x**Integer(12)*t**Integer(2) +_
↪ x**Integer(18)*t + x**Integer(21) + x**Integer(18), names=('y',)); (y,) = F.
↪_first_ngens(1)
>>> O = F.maximal_order()
>>> O.basis()
(1, 1/x^4*y, 1/x^11*y^2 + 1/x^2, 1/x^15*y^3 + 1/x^6*y)
```

number_of_rational_places(r=1)

Return the number of rational places of the function field whose constant field extended by degree r .

INPUT:

- r – positive integer (default: 1)

EXAMPLES:

```
sage: # needs sage.rings.finite_rings
sage: K.<x> = FunctionField(GF(2)); _.<Y> = K[]
sage: F.<y> = K.extension(Y^2 + Y + x + 1/x)
```

(continues on next page)

(continued from previous page)

```
sage: F.number_of_rational_places()
4
sage: [F.number_of_rational_places(r) for r in [1..10]]
[4, 8, 4, 16, 44, 56, 116, 288, 508, 968]
```

```
>>> from sage.all import *
>>> # needs sage.rings.finite_rings
>>> K = FunctionField(GF(Integer(2)), names=('x',)); (x,) = K._first_ngens(1);
↪ _ = K['Y']; (Y,) = _._first_ngens(1)
>>> F = K.extension(Y**Integer(2) + Y + x + Integer(1)/x, names=('y',)); (y,) =
↪ F._first_ngens(1)
>>> F.number_of_rational_places()
4
>>> [F.number_of_rational_places(r) for r in (ellipsis_range(Integer(1),
↪ Ellipsis, Integer(10)))]
[4, 8, 4, 16, 44, 56, 116, 288, 508, 968]
```

places(degree=1)

Return a list of the places with degree.

INPUT:

- degree – positive integer (default: 1)

EXAMPLES:

```
sage: # needs sage.rings.finite_rings
sage: F.<a> = GF(2)
sage: K.<x> = FunctionField(F)
sage: R.<t> = PolynomialRing(K)
sage: L.<y> = K.extension(t^4 + t - x^5)
sage: L.places(1)
[Place (1/x, 1/x^4*y^3), Place (x, y), Place (x, y + 1)]
```

```
>>> from sage.all import *
>>> # needs sage.rings.finite_rings
>>> F = GF(Integer(2), names=('a',)); (a,) = F._first_ngens(1)
>>> K = FunctionField(F, names=('x',)); (x,) = K._first_ngens(1)
>>> R = PolynomialRing(K, names=('t',)); (t,) = R._first_ngens(1)
>>> L = K.extension(t**Integer(4) + t - x**Integer(5), names=('y',)); (y,) =
↪ L._first_ngens(1)
>>> L.places(Integer(1))
[Place (1/x, 1/x^4*y^3), Place (x, y), Place (x, y + 1)]
```

places_finite(degree=1)

Return a list of the finite places with degree.

INPUT:

- degree – positive integer (default: 1)

EXAMPLES:

```

sage: # needs sage.rings.finite_rings
sage: F.<a> = GF(2)
sage: K.<x> = FunctionField(F)
sage: R.<t> = PolynomialRing(K)
sage: L.<y> = K.extension(t^4 + t - x^5)
sage: L.places_finite(1)
[Place (x, y), Place (x, y + 1)]

```

```

>>> from sage.all import *
>>> # needs sage.rings.finite_rings
>>> F = GF(Integer(2), names=('a',)); (a,) = F._first_ngens(1)
>>> K = FunctionField(F, names=('x',)); (x,) = K._first_ngens(1)
>>> R = PolynomialRing(K, names=('t',)); (t,) = R._first_ngens(1)
>>> L = K.extension(t**Integer(4) + t - x**Integer(5), names=('y',)); (y,) = L._first_ngens(1)
>>> L.places_finite(Integer(1))
[Place (x, y), Place (x, y + 1)]

```

weierstrass_places()

Return all Weierstrass places of the function field.

EXAMPLES:

```

sage: K.<x> = FunctionField(GF(2)); _.<Y> = K[] #_
↪ needs sage.rings.finite_rings
sage: L.<y> = K.extension(Y^3 + x^3 * Y + x) #_
↪ needs sage.rings.finite_rings
sage: L.weierstrass_places() #_
↪ needs sage.modules sage.rings.finite_rings
[Place (1/x, 1/x^3*y^2 + 1/x),
 Place (1/x, 1/x^3*y^2 + 1/x^2*y + 1),
 Place (x, y),
 Place (x + 1, (x^3 + 1)*y + x + 1),
 Place (x^3 + x + 1, y + 1),
 Place (x^3 + x + 1, y + x^2),
 Place (x^3 + x + 1, y + x^2 + 1),
 Place (x^3 + x^2 + 1, y + x),
 Place (x^3 + x^2 + 1, y + x^2 + 1),
 Place (x^3 + x^2 + 1, y + x^2 + x + 1)]

```

```

>>> from sage.all import *
>>> K = FunctionField(GF(Integer(2)), names=('x',)); (x,) = K._first_ngens(1);
↪ _ = K['Y']; (Y,) = _.first_ngens(1) # needs sage.rings.finite_rings
>>> L = K.extension(Y**Integer(3) + x**Integer(3) * Y + x, names=('y',)); (y,) = L._first_ngens(1) # needs sage.rings.finite_rings
>>> L.weierstrass_places() #_
↪ needs sage.modules sage.rings.finite_rings
[Place (1/x, 1/x^3*y^2 + 1/x),
 Place (1/x, 1/x^3*y^2 + 1/x^2*y + 1),
 Place (x, y),
 Place (x + 1, (x^3 + 1)*y + x + 1),
 Place (x^3 + x + 1, y + 1),
 Place (x^3 + x + 1, y + x^2),

```

(continues on next page)

(continued from previous page)

```
Place (x^3 + x + 1, y + x^2 + 1),
Place (x^3 + x^2 + 1, y + x),
Place (x^3 + x^2 + 1, y + x^2 + 1),
Place (x^3 + x^2 + 1, y + x^2 + x + 1)]
```

```
class sage.rings.function_field.function_field_polymod.FunctionField_global_integral (poly-
                                                    no-
                                                    mial,
                                                    names)
```

Bases: *FunctionField_global, FunctionField_integral*

Global function fields, defined by an irreducible and separable polynomial, integral over the maximal order of the base rational function field with a finite constant field.

```
class sage.rings.function_field.function_field_polymod.FunctionField_integral (polynomial,
                                                    names, cate-
                                                    gory=None)
```

Bases: *FunctionField_simple*

Integral function fields.

A function field is integral if it is defined by an irreducible separable polynomial, which is integral over the maximal order of the base rational function field.

equation_order()

Return the equation order of the function field.

EXAMPLES:

```
sage: K.<x> = FunctionField(GF(2)); R.<t> = PolynomialRing(K) #_
↪ needs sage.rings.finite_rings
sage: F.<y> = K.extension(t^3 - x^2*(x^2+x+1)^2) #_
↪ needs sage.rings.finite_rings
sage: F.equation_order() #_
↪ needs sage.rings.finite_rings
Order in Function field in y defined by y^3 + x^6 + x^4 + x^2

sage: K.<x> = FunctionField(QQ); R.<t> = PolynomialRing(K)
sage: F.<y> = K.extension(t^3 - x^2*(x^2+x+1)^2)
sage: F.equation_order()
Order in Function field in y defined by y^3 - x^6 - 2*x^5 - 3*x^4 - 2*x^3 - x^
↪ 2
```

```
>>> from sage.all import *
>>> K = FunctionField(GF(Integer(2)), names=('x',)); (x,) = K._first_ngens(1);
↪ R = PolynomialRing(K, names=('t',)); (t,) = R._first_ngens(1) # needs sage.
↪ rings.finite_rings
>>> F = K.extension(t**Integer(3) -_
↪ x**Integer(2)*(x**Integer(2)+x+Integer(1))**Integer(2), names=('y',)); (y,) =_
↪ F._first_ngens(1) # needs sage.rings.finite_rings
>>> F.equation_order() #_
↪ needs sage.rings.finite_rings
Order in Function field in y defined by y^3 + x^6 + x^4 + x^2
```

(continues on next page)

(continued from previous page)

```

>>> K = FunctionField(QQ, names=('x',)); (x,) = K._first_ngens(1); R =
↳ PolynomialRing(K, names=('t',)); (t,) = R._first_ngens(1)
>>> F = K.extension(t**Integer(3) -
↳ x**Integer(2)*(x**Integer(2)+x+Integer(1))**Integer(2), names=('y',)); (y,) =
↳ F._first_ngens(1)
>>> F.equation_order()
Order in Function field in y defined by y^3 - x^6 - 2*x^5 - 3*x^4 - 2*x^3 - x^
↳ 2

```

equation_order_infinite()

Return the infinite equation order of the function field.

This is by definition $o[b]$ where b is the primitive integral element from [primitive_integral_element_infinite\(\)](#) and o is the maximal infinite order of the base rational function field.

EXAMPLES:

```

sage: K.<x> = FunctionField(GF(2)); R.<t> = PolynomialRing(K) #_
↳ needs sage.rings.finite_rings
sage: F.<y> = K.extension(t^3 - x^2*(x^2+x+1)^2) #_
↳ needs sage.rings.finite_rings
sage: F.equation_order_infinite() #_
↳ needs sage.rings.finite_rings
Infinite order in Function field in y defined by y^3 + x^6 + x^4 + x^2

sage: K.<x> = FunctionField(QQ); R.<t> = PolynomialRing(K)
sage: F.<y> = K.extension(t^3 - x^2*(x^2+x+1)^2)
sage: F.equation_order_infinite()
Infinite order in Function field in y defined by y^3 - x^6 - 2*x^5 - 3*x^4 -
↳ 2*x^3 - x^2

```

```

>>> from sage.all import *
>>> K = FunctionField(GF(Integer(2)), names=('x',)); (x,) = K._first_ngens(1);
↳ R = PolynomialRing(K, names=('t',)); (t,) = R._first_ngens(1) # needs sage.
↳ rings.finite_rings
>>> F = K.extension(t**Integer(3) -
↳ x**Integer(2)*(x**Integer(2)+x+Integer(1))**Integer(2), names=('y',)); (y,) =
↳ F._first_ngens(1) # needs sage.rings.finite_rings
>>> F.equation_order_infinite() #_
↳ needs sage.rings.finite_rings
Infinite order in Function field in y defined by y^3 + x^6 + x^4 + x^2

>>> K = FunctionField(QQ, names=('x',)); (x,) = K._first_ngens(1); R =
↳ PolynomialRing(K, names=('t',)); (t,) = R._first_ngens(1)
>>> F = K.extension(t**Integer(3) -
↳ x**Integer(2)*(x**Integer(2)+x+Integer(1))**Integer(2), names=('y',)); (y,) =
↳ F._first_ngens(1)
>>> F.equation_order_infinite()
Infinite order in Function field in y defined by y^3 - x^6 - 2*x^5 - 3*x^4 -
↳ 2*x^3 - x^2

```

primitive_integral_element_infinite()

Return a primitive integral element over the base maximal infinite order.

This element is integral over the maximal infinite order of the base rational function field and the function field is a simple extension by this element over the base order.

EXAMPLES:

```
sage: # needs sage.rings.finite_rings
sage: K.<x> = FunctionField(GF(2)); R.<t> = PolynomialRing(K)
sage: F.<y> = K.extension(t^3 - x^2*(x^2+x+1)^2)
sage: b = F.primitive_integral_element_infinite(); b
1/x^2*y
sage: b.minimal_polynomial('t')
t^3 + (x^4 + x^2 + 1)/x^4
```

```
>>> from sage.all import *
>>> # needs sage.rings.finite_rings
>>> K = FunctionField(GF(Integer(2)), names=('x',)); (x,) = K._first_ngens(1);
↳ R = PolynomialRing(K, names=('t',)); (t,) = R._first_ngens(1)
>>> F = K.extension(t**Integer(3) -
↳ x**Integer(2)*(x**Integer(2)+x+Integer(1))**Integer(2), names=('y',)); (y,) =
↳ F._first_ngens(1)
>>> b = F.primitive_integral_element_infinite(); b
1/x^2*y
>>> b.minimal_polynomial('t')
t^3 + (x^4 + x^2 + 1)/x^4
```

```
class sage.rings.function_field.function_field_polymod.FunctionField_polymod(polynomial,
                                                                              names, cate-
                                                                              gory=None)
```

Bases: *FunctionField*

Function fields defined by a univariate polynomial, as an extension of the base field.

INPUT:

- *polynomial* – univariate polynomial over a function field
- *names* – tuple of length 1 or string; variable names
- *category* – category (default: category of function fields)

EXAMPLES:

We make a function field defined by a degree 5 polynomial over the rational function field over the rational numbers:

```
sage: K.<x> = FunctionField(QQ)
sage: R.<y> = K[]
sage: L.<y> = K.extension(y^5 - (x^3 + 2*x*y + 1/x)); L
Function field in y defined by y^5 - 2*x*y + (-x^4 - 1)/x
```

```
>>> from sage.all import *
>>> K = FunctionField(QQ, names=('x',)); (x,) = K._first_ngens(1)
>>> R = K['y']; (y,) = R._first_ngens(1)
>>> L = K.extension(y**Integer(5) - (x**Integer(3) + Integer(2)*x*y + Integer(1)/
↳ x), names=('y',)); (y,) = L._first_ngens(1); L
Function field in y defined by y^5 - 2*x*y + (-x^4 - 1)/x
```

We next make a function field over the above nontrivial function field L:

```

sage: S.<z> = L[]
sage: M.<z> = L.extension(z^2 + y*z + y); M
Function field in z defined by z^2 + y*z + y
sage: 1/z
((-x/(x^4 + 1))*y^4 + 2*x^2/(x^4 + 1))*z - 1
sage: z * (1/z)
1

```

```

>>> from sage.all import *
>>> S = L['z']; (z,) = S._first_ngens(1)
>>> M = L.extension(z**Integer(2) + y*z + y, names=('z',)); (z,) = M._first_
↳ngens(1); M
Function field in z defined by z^2 + y*z + y
>>> Integer(1)/z
((-x/(x^4 + 1))*y^4 + 2*x^2/(x^4 + 1))*z - 1
>>> z * (Integer(1)/z)
1

```

We drill down the tower of function fields:

```

sage: M.base_field()
Function field in y defined by y^5 - 2*x*y + (-x^4 - 1)/x
sage: M.base_field().base_field()
Rational function field in x over Rational Field
sage: M.base_field().base_field().constant_field()
Rational Field
sage: M.constant_base_field()
Rational Field

```

```

>>> from sage.all import *
>>> M.base_field()
Function field in y defined by y^5 - 2*x*y + (-x^4 - 1)/x
>>> M.base_field().base_field()
Rational function field in x over Rational Field
>>> M.base_field().base_field().constant_field()
Rational Field
>>> M.constant_base_field()
Rational Field

```

Warning

It is not checked if the polynomial used to define the function field is irreducible. Hence it is not guaranteed that this object really is a field! This is illustrated below.

```

sage: K.<x> = FunctionField(QQ)
sage: R.<y> = K[]
sage: L.<y> = K.extension(x^2 - y^2)
sage: (y - x)*(y + x)
0
sage: 1/(y - x)

```

(continues on next page)

(continued from previous page)

```
1
sage: y - x == 0; y + x == 0
False
False
```

```
>>> from sage.all import *
>>> K = FunctionField(QQ, names=('x',)); (x,) = K._first_ngens(1)
>>> R = K['y']; (y,) = R._first_ngens(1)
>>> L = K.extension(x**Integer(2) - y**Integer(2), names=('y',)); (y,) = L._first_
↳ngens(1)
>>> (y - x)*(y + x)
0
>>> Integer(1)/(y - x)
1
>>> y - x == Integer(0); y + x == Integer(0)
False
False
```

Elementalias of *FunctionFieldElement_polymod***base_field()**

Return the base field of the function field. This function field is presented as $L = K[y]/(f(y))$, and the base field is by definition the field K .

EXAMPLES:

```
sage: K.<x> = FunctionField(QQ); R.<y> = K[]
sage: L.<y> = K.extension(y^5 - (x^3 + 2*x*y + 1/x))
sage: L.base_field()
Rational function field in x over Rational Field
```

```
>>> from sage.all import *
>>> K = FunctionField(QQ, names=('x',)); (x,) = K._first_ngens(1); R = K['y'];
↳ (y,) = R._first_ngens(1)
>>> L = K.extension(y**Integer(5) - (x**Integer(3) + Integer(2)*x*y +
↳Integer(1)/x), names=('y',)); (y,) = L._first_ngens(1)
>>> L.base_field()
Rational function field in x over Rational Field
```

change_variable_name (name)

Return a field isomorphic to this field with variable(s) name.

INPUT:

- name – string or tuple consisting of a strings; the names of the new variables starting with a generator of this field and going down to the rational function field

OUTPUT:

A triple F, f, t where F is a function field, f is an isomorphism from F to this field, and t is the inverse of f .

EXAMPLES:

```

sage: K.<x> = FunctionField(QQ)
sage: R.<y> = K[]
sage: L.<y> = K.extension(y^2 - x)
sage: R.<z> = L[]
sage: M.<z> = L.extension(z^2 - y)

sage: M.change_variable_name('zz')
(Function field in zz defined by zz^2 - y,
Function Field morphism:
  From: Function field in zz defined by zz^2 - y
  To:   Function field in z defined by z^2 - y
  Defn: zz |--> z
        y |--> y
        x |--> x,
Function Field morphism:
  From: Function field in z defined by z^2 - y
  To:   Function field in zz defined by zz^2 - y
  Defn: z |--> zz
        y |--> y
        x |--> x)

sage: M.change_variable_name(('zz', 'yy'))
(Function field in zz defined by zz^2 - yy,
Function Field morphism:
  From: Function field in zz defined by zz^2 - yy
  To:   Function field in z defined by z^2 - y
  Defn: zz |--> z
        yy |--> y
        x |--> x,
Function Field morphism:
  From: Function field in z defined by z^2 - y
  To:   Function field in zz defined by zz^2 - yy
  Defn: z |--> zz
        y |--> yy
        x |--> x)

sage: M.change_variable_name(('zz', 'yy', 'xx'))
(Function field in zz defined by zz^2 - yy,
Function Field morphism:
  From: Function field in zz defined by zz^2 - yy
  To:   Function field in z defined by z^2 - y
  Defn: zz |--> z
        yy |--> y
        xx |--> x,
Function Field morphism:
  From: Function field in z defined by z^2 - y
  To:   Function field in zz defined by zz^2 - yy
  Defn: z |--> zz
        y |--> yy
        x |--> xx)

```

```

>>> from sage.all import *
>>> K = FunctionField(QQ, names=('x',)); (x,) = K._first_ngens(1)
>>> R = K['y']; (y,) = R._first_ngens(1)

```

(continues on next page)

(continued from previous page)

```

>>> L = K.extension(y**Integer(2) - x, names=('y',)); (y,) = L._first_ngens(1)
>>> R = L['z']; (z,) = R._first_ngens(1)
>>> M = L.extension(z**Integer(2) - y, names=('z',)); (z,) = M._first_ngens(1)

>>> M.change_variable_name('zz')
(Function field in zz defined by zz^2 - y,
Function Field morphism:
  From: Function field in zz defined by zz^2 - y
  To:   Function field in z defined by z^2 - y
  Defn: zz |--> z
        y |--> y
        x |--> x,
Function Field morphism:
  From: Function field in z defined by z^2 - y
  To:   Function field in zz defined by zz^2 - y
  Defn: z |--> zz
        y |--> y
        x |--> x)

>>> M.change_variable_name(('zz','yy'))
(Function field in zz defined by zz^2 - yy,
Function Field morphism:
  From: Function field in zz defined by zz^2 - yy
  To:   Function field in z defined by z^2 - y
  Defn: zz |--> z
        yy |--> y
        x |--> x,
Function Field morphism:
  From: Function field in z defined by z^2 - y
  To:   Function field in zz defined by zz^2 - yy
  Defn: z |--> zz
        y |--> yy
        x |--> x)

>>> M.change_variable_name(('zz','yy','xx'))
(Function field in zz defined by zz^2 - yy,
Function Field morphism:
  From: Function field in zz defined by zz^2 - yy
  To:   Function field in z defined by z^2 - y
  Defn: zz |--> z
        yy |--> y
        xx |--> x,
Function Field morphism:
  From: Function field in z defined by z^2 - y
  To:   Function field in zz defined by zz^2 - yy
  Defn: z |--> zz
        y |--> yy
        x |--> xx)

```

constant_base_field()

Return the base constant field of the function field.

EXAMPLES:

```

sage: K.<x> = FunctionField(QQ); R.<y> = K[]
sage: L.<y> = K.extension(y^5 - (x^3 + 2*x*y + 1/x)); L
Function field in y defined by y^5 - 2*x*y + (-x^4 - 1)/x
sage: L.constant_base_field()
Rational Field
sage: S.<z> = L[]
sage: M.<z> = L.extension(z^2 - y)
sage: M.constant_base_field()
Rational Field

```

```

>>> from sage.all import *
>>> K = FunctionField(QQ, names=('x',)); (x,) = K._first_ngens(1); R = K['y'];
↪ (y,) = R._first_ngens(1)
>>> L = K.extension(y**Integer(5) - (x**Integer(3) + Integer(2)*x*y +
↪ Integer(1)/x), names=('y',)); (y,) = L._first_ngens(1); L
Function field in y defined by y^5 - 2*x*y + (-x^4 - 1)/x
>>> L.constant_base_field()
Rational Field
>>> S = L['z']; (z,) = S._first_ngens(1)
>>> M = L.extension(z**Integer(2) - y, names=('z',)); (z,) = M._first_ngens(1)
>>> M.constant_base_field()
Rational Field

```

constant_field()

Return the algebraic closure of the constant field of the function field.

EXAMPLES:

```

sage: K.<x> = FunctionField(GF(5)); _.<Y> = K[] #_
↪ needs sage.rings.finite_rings
sage: L.<y> = K.extension(Y^5 - x) #_
↪ needs sage.rings.finite_rings
sage: L.constant_field() #_
↪ needs sage.rings.finite_rings
Traceback (most recent call last):
...
NotImplementedError

```

```

>>> from sage.all import *
>>> K = FunctionField(GF(Integer(5)), names=('x',)); (x,) = K._first_ngens(1);
↪ _ = K['Y']; (Y,) = _. _first_ngens(1) # needs sage.rings.finite_rings
>>> L = K.extension(Y**Integer(5) - x, names=('y',)); (y,) = L._first_ngens(1)
↪ # needs sage.rings.finite_rings
>>> L.constant_field() #_
↪ needs sage.rings.finite_rings
Traceback (most recent call last):
...
NotImplementedError

```

degree (base=None)

Return the degree of the function field over the function field base.

INPUT:

- `base` – a function field (default: `None`), a function field from which this field has been constructed as a finite extension

EXAMPLES:

```
sage: K.<x> = FunctionField(QQ)
sage: R.<y> = K[]
sage: L.<y> = K.extension(y^5 - (x^3 + 2*x*y + 1/x)); L
Function field in y defined by y^5 - 2*x*y + (-x^4 - 1)/x
sage: L.degree()
5
sage: L.degree(L)
1

sage: R.<z> = L[]
sage: M.<z> = L.extension(z^2 - y)
sage: M.degree(L)
2
sage: M.degree(K)
10
```

```
>>> from sage.all import *
>>> K = FunctionField(QQ, names=('x',)); (x,) = K._first_ngens(1)
>>> R = K['y']; (y,) = R._first_ngens(1)
>>> L = K.extension(y**Integer(5) - (x**Integer(3) + Integer(2)*x*y +
↳ Integer(1)/x), names=('y',)); (y,) = L._first_ngens(1); L
Function field in y defined by y^5 - 2*x*y + (-x^4 - 1)/x
>>> L.degree()
5
>>> L.degree(L)
1

>>> R = L['z']; (z,) = R._first_ngens(1)
>>> M = L.extension(z**Integer(2) - y, names=('z',)); (z,) = M._first_ngens(1)
>>> M.degree(L)
2
>>> M.degree(K)
10
```

different()

Return the different of the function field.

EXAMPLES:

```
sage: K.<x> = FunctionField(GF(2)); _.<Y> = K[] #_
↳ needs sage.rings.finite_rings
sage: F.<y> = K.extension(Y^3 - x^2*(x^2 + x + 1)^2) #_
↳ needs sage.rings.finite_rings
sage: F.different() #_
↳ needs sage.rings.finite_rings
2*Place (x, (1/(x^3 + x^2 + x))*y^2)
+ 2*Place (x^2 + x + 1, (1/(x^3 + x^2 + x))*y^2)
```

```

>>> from sage.all import *
>>> K = FunctionField(GF(Integer(2)), names=('x',)); (x,) = K._first_ngens(1);
↪ _ = K['Y']; (Y,) = _._first_ngens(1) # needs sage.rings.finite_rings
>>> F = K.extension(Y**Integer(3) - x**Integer(2)*(x**Integer(2) + x +
↪ Integer(1))**Integer(2), names=('y',)); (y,) = F._first_ngens(1) # needs
↪ sage.rings.finite_rings
>>> F.different() #
↪ needs sage.rings.finite_rings
2*Place (x, (1/(x^3 + x^2 + x))*y^2)
+ 2*Place (x^2 + x + 1, (1/(x^3 + x^2 + x))*y^2)

```

equation_order()

Return the equation order of the function field.

If we view the function field as being presented as $K[y]/(f(y))$, then the order generated by the class of y is returned. If f is not monic, then `_make_monic_integral()` is called, and instead we get the order generated by some integral multiple of a root of f .

EXAMPLES:

```

sage: K.<x> = FunctionField(QQ); R.<y> = K[]
sage: L.<y> = K.extension(y^5 - (x^3 + 2*x*y + 1/x))
sage: O = L.equation_order()
sage: O.basis()
(1, x*y, x^2*y^2, x^3*y^3, x^4*y^4)

```

```

>>> from sage.all import *
>>> K = FunctionField(QQ, names=('x',)); (x,) = K._first_ngens(1); R = K['y'];
↪ (y,) = R._first_ngens(1)
>>> L = K.extension(y**Integer(5) - (x**Integer(3) + Integer(2)*x*y +
↪ Integer(1)/x), names=('y',)); (y,) = L._first_ngens(1)
>>> O = L.equation_order()
>>> O.basis()
(1, x*y, x^2*y^2, x^3*y^3, x^4*y^4)

```

We try an example, in which the defining polynomial is not monic and is not integral:

```

sage: K.<x> = FunctionField(QQ); R.<y> = K[]
sage: L.<y> = K.extension(x^2*y^5 - 1/x); L
Function field in y defined by x^2*y^5 - 1/x
sage: O = L.equation_order()
sage: O.basis()
(1, x^3*y, x^6*y^2, x^9*y^3, x^12*y^4)

```

```

>>> from sage.all import *
>>> K = FunctionField(QQ, names=('x',)); (x,) = K._first_ngens(1); R = K['y'];
↪ (y,) = R._first_ngens(1)
>>> L = K.extension(x**Integer(2)*y**Integer(5) - Integer(1)/x, names=('y',));
↪ (y,) = L._first_ngens(1); L
Function field in y defined by x^2*y^5 - 1/x
>>> O = L.equation_order()
>>> O.basis()
(1, x^3*y, x^6*y^2, x^9*y^3, x^12*y^4)

```


free_module (*base=None, basis=None, map=True*)

Return a vector space and isomorphisms from the field to and from the vector space.

This function allows us to identify the elements of this field with elements of a vector space over the base field, which is useful for representation and arithmetic with orders, ideals, etc.

INPUT:

- *base* – a function field (default: *None*), the returned vector space is over this subfield R , which defaults to the base field of this function field.
- *basis* – a basis for this field over the base
- *maps* – boolean (default: *True*); whether to return R -linear maps to and from V

OUTPUT:

- a vector space over the base function field
- an isomorphism from the vector space to the field (if requested)
- an isomorphism from the field to the vector space (if requested)

EXAMPLES:

We define a function field:

```
sage: K.<x> = FunctionField(QQ); R.<y> = K[]
sage: L.<y> = K.extension(y^5 - (x^3 + 2*x*y + 1/x)); L
Function field in y defined by y^5 - 2*x*y + (-x^4 - 1)/x
```

```
>>> from sage.all import *
>>> K = FunctionField(QQ, names=('x',)); (x,) = K._first_ngens(1); R = K['y'];
↪ (y,) = R._first_ngens(1)
>>> L = K.extension(y**Integer(5) - (x**Integer(3) + Integer(2)*x*y +
↪ Integer(1)/x), names=('y',)); (y,) = L._first_ngens(1); L
Function field in y defined by y^5 - 2*x*y + (-x^4 - 1)/x
```

We get the vector spaces, and maps back and forth:

```
sage: # needs sage.modules
sage: V, from_V, to_V = L.free_module()
sage: V
Vector space of dimension 5 over Rational function field in x over Rational_
↪Field
sage: from_V
Isomorphism:
  From: Vector space of dimension 5 over Rational function field in x over_
↪Rational Field
  To:   Function field in y defined by y^5 - 2*x*y + (-x^4 - 1)/x
sage: to_V
Isomorphism:
  From: Function field in y defined by y^5 - 2*x*y + (-x^4 - 1)/x
  To:   Vector space of dimension 5 over Rational function field in x over_
↪Rational Field
```

```
>>> from sage.all import *
>>> # needs sage.modules
```

(continues on next page)

(continued from previous page)

```

>>> V, from_V, to_V = L.free_module()
>>> V
Vector space of dimension 5 over Rational function field in x over Rational
↳Field
>>> from_V
Isomorphism:
  From: Vector space of dimension 5 over Rational function field in x over
↳Rational Field
  To:   Function field in y defined by  $y^5 - 2xy + (-x^4 - 1)/x$ 
>>> to_V
Isomorphism:
  From: Function field in y defined by  $y^5 - 2xy + (-x^4 - 1)/x$ 
  To:   Vector space of dimension 5 over Rational function field in x over
↳Rational Field

```

We convert an element of the vector space back to the function field:

```

sage: from_V(V.1)                                     #↳
↳needs sage.modules
y

```

```

>>> from sage.all import *
>>> from_V(V.gen(1))                                  ↳
↳ # needs sage.modules
y

```

We define an interesting element of the function field:

```

sage: a = 1/L.0; a                                     #↳
↳needs sage.modules
(x/(x^4 + 1))*y^4 - 2*x^2/(x^4 + 1)

```

```

>>> from sage.all import *
>>> a = Integer(1)/L.gen(0); a                         ↳
↳ # needs sage.modules
(x/(x^4 + 1))*y^4 - 2*x^2/(x^4 + 1)

```

We convert it to the vector space, and get a vector over the base field:

```

sage: to_V(a)                                           #↳
↳needs sage.modules
(-2*x^2/(x^4 + 1), 0, 0, 0, x/(x^4 + 1))

```

```

>>> from sage.all import *
>>> to_V(a)                                             #↳
↳needs sage.modules
(-2*x^2/(x^4 + 1), 0, 0, 0, x/(x^4 + 1))

```

We convert to and back, and get the same element:

```

sage: from_V(to_V(a)) == a                             #↳
↳needs sage.modules

```

(continues on next page)

(continued from previous page)

True

```
>>> from sage.all import *
>>> from_V(to_V(a)) == a #_
↳needs sage.modules
True
```

In the other direction:

```
sage: v = x*V.0 + (1/x)*V.1 #_
↳needs sage.modules
sage: to_V(from_V(v)) == v #_
↳needs sage.modules
True
```

```
>>> from sage.all import *
>>> v = x*V.gen(0) + (Integer(1)/x)*V.gen(1) #_
↳ # needs sage.modules
>>> to_V(from_V(v)) == v #_
↳needs sage.modules
True
```

And we show how it works over an extension of an extension field:

```
sage: R2.<z> = L[]; M.<z> = L.extension(z^2 - y)
sage: M.free_module() #_
↳needs sage.modules
(Vector space of dimension 2 over Function field in y defined by y^5 - 2*x*y
↳+ (-x^4 - 1)/x,
Isomorphism:
From: Vector space of dimension 2 over Function field in y defined by y^5 -
↳2*x*y + (-x^4 - 1)/x
To: Function field in z defined by z^2 - y,
Isomorphism:
From: Function field in z defined by z^2 - y
To: Vector space of dimension 2 over Function field in y defined by y^5 -
↳2*x*y + (-x^4 - 1)/x)
```

```
>>> from sage.all import *
>>> R2 = L['z']; (z,) = R2._first_ngens(1); M = L.extension(z**Integer(2) - y,
↳ names=('z',)); (z,) = M._first_ngens(1)
>>> M.free_module() #_
↳needs sage.modules
(Vector space of dimension 2 over Function field in y defined by y^5 - 2*x*y
↳+ (-x^4 - 1)/x,
Isomorphism:
From: Vector space of dimension 2 over Function field in y defined by y^5 -
↳2*x*y + (-x^4 - 1)/x
To: Function field in z defined by z^2 - y,
Isomorphism:
From: Function field in z defined by z^2 - y
```

(continues on next page)

(continued from previous page)

```
To: Vector space of dimension 2 over Function field in y defined by y^5 -
↳ 2*x*y + (-x^4 - 1)/x)
```

We can also get the vector space of M over K :

```
sage: M.free_module(K) #
↳ needs sage.modules
(Vector space of dimension 10 over Rational function field in x over Rational
↳ Field,
Isomorphism:
From: Vector space of dimension 10 over Rational function field in x over
↳ Rational Field
To: Function field in z defined by z^2 - y,
Isomorphism:
From: Function field in z defined by z^2 - y
To: Vector space of dimension 10 over Rational function field in x over
↳ Rational Field)
```

```
>>> from sage.all import *
>>> M.free_module(K) #
↳ needs sage.modules
(Vector space of dimension 10 over Rational function field in x over Rational
↳ Field,
Isomorphism:
From: Vector space of dimension 10 over Rational function field in x over
↳ Rational Field
To: Function field in z defined by z^2 - y,
Isomorphism:
From: Function field in z defined by z^2 - y
To: Vector space of dimension 10 over Rational function field in x over
↳ Rational Field)
```

gen ($n=0$)

Return the n -th generator of the function field. By default, n is 0; any other value of n leads to an error. The generator is the class of y , if we view the function field as being presented as $K[y]/(f(y))$.

EXAMPLES:

```
sage: K.<x> = FunctionField(QQ); R.<y> = K[]
sage: L.<y> = K.extension(y^5 - (x^3 + 2*x*y + 1/x))
sage: L.gen()
y
sage: L.gen(1)
Traceback (most recent call last):
...
IndexError: there is only one generator
```

```
>>> from sage.all import *
>>> K = FunctionField(QQ, names=('x',)); (x,) = K._first_ngens(1); R = K['y'];
↳ (y,) = R._first_ngens(1)
>>> L = K.extension(y**Integer(5) - (x**Integer(3) + Integer(2)*x*y +
↳ Integer(1)/x), names=('y',)); (y,) = L._first_ngens(1)
```

(continues on next page)

(continued from previous page)

```
>>> L.gen()
y
>>> L.gen(Integer(1))
Traceback (most recent call last):
...
IndexError: there is only one generator
```

genus()

Return the genus of the function field.

For now, the genus is computed using Singular.

EXAMPLES:

```
sage: K.<x> = FunctionField(QQ); R.<y> = K[]
sage: L.<y> = K.extension(y^3 - (x^3 + 2*x*y + 1/x))
sage: L.genus()
3
```

```
>>> from sage.all import *
>>> K = FunctionField(QQ, names=('x',)); (x,) = K._first_ngens(1); R = K['y'];
↪ (y,) = R._first_ngens(1)
>>> L = K.extension(y**Integer(3) - (x**Integer(3) + Integer(2)*x*y +
↪ Integer(1)/x), names=('y',)); (y,) = L._first_ngens(1)
>>> L.genus()
3
```

hom(im_gens, base_morphism=None)

Create a homomorphism from the function field to another function field.

INPUT:

- `im_gens` – list of images of the generators of the function field and of successive base rings
- `base_morphism` – homomorphism of the base ring, after the `im_gens` are used. Thus if `im_gens` has length 2, then `base_morphism` should be a morphism from the base ring of the base ring of the function field.

EXAMPLES:

We create a rational function field, and a quadratic extension of it:

```
sage: K.<x> = FunctionField(QQ); R.<y> = K[]
sage: L.<y> = K.extension(y^2 - x^3 - 1)
```

```
>>> from sage.all import *
>>> K = FunctionField(QQ, names=('x',)); (x,) = K._first_ngens(1); R = K['y'];
↪ (y,) = R._first_ngens(1)
>>> L = K.extension(y**Integer(2) - x**Integer(3) - Integer(1), names=('y',));
↪ (y,) = L._first_ngens(1)
```

We make the field automorphism that sends `y` to `-y`:

```
sage: f = L.hom(-y); f
Function Field endomorphism of Function field in y defined by y^2 - x^3 - 1
Defn: y |--> -y
```

```
>>> from sage.all import *
>>> f = L.hom(-y); f
Function Field endomorphism of Function field in y defined by  $y^2 - x^3 - 1$ 
Defn:  $y \mapsto -y$ 
```

Evaluation works:

```
sage: f(y*x - 1/x)
-x*y - 1/x
```

```
>>> from sage.all import *
>>> f(y*x - Integer(1)/x)
-x*y - 1/x
```

We try to define an invalid morphism:

```
sage: f = L.hom(y + 1)
Traceback (most recent call last):
...
ValueError: invalid morphism
```

```
>>> from sage.all import *
>>> f = L.hom(y + Integer(1))
Traceback (most recent call last):
...
ValueError: invalid morphism
```

We make a morphism of the base rational function field:

```
sage: phi = K.hom(x + 1); phi
Function Field endomorphism of Rational function field in x over Rational_
↪Field
Defn:  $x \mapsto x + 1$ 
sage: phi(x^3 - 3)
 $x^3 + 3x^2 + 3x - 2$ 
sage: (x+1)^3 - 3
 $x^3 + 3x^2 + 3x - 2$ 
```

```
>>> from sage.all import *
>>> phi = K.hom(x + Integer(1)); phi
Function Field endomorphism of Rational function field in x over Rational_
↪Field
Defn:  $x \mapsto x + 1$ 
>>> phi(x**Integer(3) - Integer(3))
 $x^3 + 3x^2 + 3x - 2$ 
>>> (x+Integer(1))**Integer(3) - Integer(3)
 $x^3 + 3x^2 + 3x - 2$ 
```

We make a morphism by specifying where the generators and the base generators go:

```
sage: L.hom([-y, x])
Function Field endomorphism of Function field in y defined by  $y^2 - x^3 - 1$ 
```

(continues on next page)

(continued from previous page)

```
Defn: y |--> -y
      x |--> x
```

```
>>> from sage.all import *
>>> L.hom([-y, x])
Function Field endomorphism of Function field in y defined by y^2 - x^3 - 1
Defn: y |--> -y
      x |--> x
```

You can also specify a morphism on the base:

```
sage: R1.<q> = K[]
sage: L1.<q> = K.extension(q^2 - (x+1)^3 - 1)
sage: L.hom(q, base_morphism=phi)
Function Field morphism:
  From: Function field in y defined by y^2 - x^3 - 1
  To:   Function field in q defined by q^2 - x^3 - 3*x^2 - 3*x - 2
Defn: y |--> q
      x |--> x + 1
```

```
>>> from sage.all import *
>>> R1 = K['q']; (q,) = R1._first_ngens(1)
>>> L1 = K.extension(q**Integer(2) - (x+Integer(1))**Integer(3) - Integer(1),
↳ names=('q',)); (q,) = L1._first_ngens(1)
>>> L.hom(q, base_morphism=phi)
Function Field morphism:
  From: Function field in y defined by y^2 - x^3 - 1
  To:   Function field in q defined by q^2 - x^3 - 3*x^2 - 3*x - 2
Defn: y |--> q
      x |--> x + 1
```

We make another extension of a rational function field:

```
sage: K2.<t> = FunctionField(QQ); R2.<w> = K2[]
sage: L2.<w> = K2.extension((4*w)^2 - (t+1)^3 - 1)
```

```
>>> from sage.all import *
>>> K2 = FunctionField(QQ, names=('t',)); (t,) = K2._first_ngens(1); R2 = K2[
↳ 'w']; (w,) = R2._first_ngens(1)
>>> L2 = K2.extension((Integer(4)*w)**Integer(2) - (t+Integer(1))**Integer(3) -
↳ Integer(1), names=('w',)); (w,) = L2._first_ngens(1)
```

We define a morphism, by giving the images of generators:

```
sage: f = L.hom([4*w, t + 1]); f
Function Field morphism:
  From: Function field in y defined by y^2 - x^3 - 1
  To:   Function field in w defined by 16*w^2 - t^3 - 3*t^2 - 3*t - 2
Defn: y |--> 4*w
      x |--> t + 1
```

```
>>> from sage.all import *
>>> f = L.hom([Integer(4)*w, t + Integer(1)]); f
Function Field morphism:
  From: Function field in y defined by  $y^2 - x^3 - 1$ 
  To:   Function field in w defined by  $16w^2 - t^3 - 3t^2 - 3t - 2$ 
  Defn:  $y \mapsto 4w$ 
         $x \mapsto t + 1$ 
```

Evaluation works, as expected:

```
sage: f(y+x)
4*w + t + 1
sage: f(x*y + x/(x^2+1))
(4*t + 4)*w + (t + 1)/(t^2 + 2*t + 2)
```

```
>>> from sage.all import *
>>> f(y+x)
4*w + t + 1
>>> f(x*y + x/(x**Integer(2)+Integer(1)))
(4*t + 4)*w + (t + 1)/(t^2 + 2*t + 2)
```

We make another extension of a rational function field:

```
sage: K3.<yy> = FunctionField(QQ); R3.<xx> = K3[]
sage: L3.<xx> = K3.extension(yy^2 - xx^3 - 1)
```

```
>>> from sage.all import *
>>> K3 = FunctionField(QQ, names=('yy',)); (yy,) = K3._first_ngens(1); R3 = K3['xx']; (xx,) = R3._first_ngens(1)
>>> L3 = K3.extension(yy**Integer(2) - xx**Integer(3) - Integer(1), names=('xx',)); (xx,) = L3._first_ngens(1)
```

This is the function field L with the generators exchanged. We define a morphism to L:

```
sage: g = L3.hom([x,y]); g
Function Field morphism:
  From: Function field in xx defined by  $-xx^3 + yy^2 - 1$ 
  To:   Function field in y defined by  $y^2 - x^3 - 1$ 
  Defn:  $xx \mapsto x$ 
         $yy \mapsto y$ 
```

```
>>> from sage.all import *
>>> g = L3.hom([x,y]); g
Function Field morphism:
  From: Function field in xx defined by  $-xx^3 + yy^2 - 1$ 
  To:   Function field in y defined by  $y^2 - x^3 - 1$ 
  Defn:  $xx \mapsto x$ 
         $yy \mapsto y$ 
```

is_separable (base=None)

Return whether this is a separable extension of base.

INPUT:

- `base` – a function field from which this field has been created as an extension or `None` (default: `None`); if `None`, then return whether this is a separable extension over its base field.

EXAMPLES:

```
sage: # needs sage.rings.finite_rings
sage: K.<x> = FunctionField(GF(2))
sage: R.<y> = K[]
sage: L.<y> = K.extension(y^2 - x)
sage: L.is_separable()
False
sage: R.<z> = L[]
sage: M.<z> = L.extension(z^3 - y)
sage: M.is_separable()
True
sage: M.is_separable(K)
False

sage: # needs sage.rings.finite_rings
sage: K.<x> = FunctionField(GF(5))
sage: R.<y> = K[]
sage: L.<y> = K.extension(y^5 - (x^3 + 2*x*y + 1/x))
sage: L.is_separable()
True

sage: # needs sage.rings.finite_rings
sage: K.<x> = FunctionField(GF(5))
sage: R.<y> = K[]
sage: L.<y> = K.extension(y^5 - 1)
sage: L.is_separable()
False
```

```
>>> from sage.all import *
>>> # needs sage.rings.finite_rings
>>> K = FunctionField(GF(Integer(2)), names=('x',)); (x,) = K._first_ngens(1)
>>> R = K['y']; (y,) = R._first_ngens(1)
>>> L = K.extension(y**Integer(2) - x, names=('y',)); (y,) = L._first_ngens(1)
>>> L.is_separable()
False
>>> R = L['z']; (z,) = R._first_ngens(1)
>>> M = L.extension(z**Integer(3) - y, names=('z',)); (z,) = M._first_ngens(1)
>>> M.is_separable()
True
>>> M.is_separable(K)
False

>>> # needs sage.rings.finite_rings
>>> K = FunctionField(GF(Integer(5)), names=('x',)); (x,) = K._first_ngens(1)
>>> R = K['y']; (y,) = R._first_ngens(1)
>>> L = K.extension(y**Integer(5) - (x**Integer(3) + Integer(2)*x*y +
↳ Integer(1)/x), names=('y',)); (y,) = L._first_ngens(1)
>>> L.is_separable()
True
```

(continues on next page)

(continued from previous page)

```

>>> # needs sage.rings.finite_rings
>>> K = FunctionField(GF(Integer(5)), names=('x',)); (x,) = K._first_ngens(1)
>>> R = K['y']; (y,) = R._first_ngens(1)
>>> L = K.extension(y**Integer(5) - Integer(1), names=('y',)); (y,) = L._
↳first_ngens(1)
>>> L.is_separable()
False

```

maximal_order()

Return the maximal order of the function field.

EXAMPLES:

```

sage: K.<x> = FunctionField(QQ); R.<y> = K[]
sage: L.<y> = K.extension(y^5 - (x^3 + 2*x*y + 1/x))
sage: L.maximal_order()
Maximal order of Function field in y defined by  $y^5 - 2xy + (-x^4 - 1)/x$ 

```

```

>>> from sage.all import *
>>> K = FunctionField(QQ, names=('x',)); (x,) = K._first_ngens(1); R = K['y'];
↳(y,) = R._first_ngens(1)
>>> L = K.extension(y**Integer(5) - (x**Integer(3) + Integer(2)*x*y +
↳Integer(1)/x), names=('y',)); (y,) = L._first_ngens(1)
>>> L.maximal_order()
Maximal order of Function field in y defined by  $y^5 - 2xy + (-x^4 - 1)/x$ 

```

maximal_order_infinite()

Return the maximal infinite order of the function field.

EXAMPLES:

```

sage: K.<x> = FunctionField(QQ); R.<y> = K[]
sage: L.<y> = K.extension(y^5 - (x^3 + 2*x*y + 1/x))
sage: L.maximal_order_infinite()
Maximal infinite order of Function field in y defined by  $y^5 - 2xy + (-x^4 - 1)/x$ 

sage: K.<x> = FunctionField(GF(2)); _.<t> = K[] #_
↳needs sage.rings.finite_rings
sage: F.<y> = K.extension(t^3 - x^2*(x^2 + x + 1)^2) #_
↳needs sage.rings.finite_rings
sage: F.maximal_order_infinite() #_
↳needs sage.rings.finite_rings
Maximal infinite order of Function field in y defined by  $y^3 + x^6 + x^4 + x^2$ 

sage: K.<x> = FunctionField(GF(2)); _.<Y> = K[] #_
↳needs sage.rings.finite_rings
sage: L.<y> = K.extension(Y^2 + Y + x + 1/x) #_
↳needs sage.rings.finite_rings
sage: L.maximal_order_infinite() #_
↳needs sage.rings.finite_rings
Maximal infinite order of Function field in y defined by  $y^2 + y + (x^2 + 1)/x$ 

```

```

>>> from sage.all import *
>>> K = FunctionField(QQ, names=('x',)); (x,) = K._first_ngens(1); R = K['y'];
↳ (y,) = R._first_ngens(1)
>>> L = K.extension(y**Integer(5) - (x**Integer(3) + Integer(2)*x*y +
↳ Integer(1)/x), names=('y',)); (y,) = L._first_ngens(1)
>>> L.maximal_order_infinite()
Maximal infinite order of Function field in y defined by  $y^5 - 2xy + (-x^4 -$ 
↳  $1)/x$ 

>>> K = FunctionField(GF(Integer(2)), names=('x',)); (x,) = K._first_ngens(1);
↳ _ = K['t']; (t,) = _._first_ngens(1) # needs sage.rings.finite_rings
>>> F = K.extension(t**Integer(3) - x**Integer(2)*(x**Integer(2) + x +
↳ Integer(1))**Integer(2), names=('y',)); (y,) = F._first_ngens(1) # needs
↳ sage.rings.finite_rings
>>> F.maximal_order_infinite() #
↳ needs sage.rings.finite_rings
Maximal infinite order of Function field in y defined by  $y^3 + x^6 + x^4 + x^2$ 

>>> K = FunctionField(GF(Integer(2)), names=('x',)); (x,) = K._first_ngens(1);
↳ _ = K['Y']; (Y,) = _._first_ngens(1) # needs sage.rings.finite_rings
>>> L = K.extension(Y**Integer(2) + Y + x + Integer(1)/x, names=('y',)); (y,)
↳ = L._first_ngens(1) # needs sage.rings.finite_rings
>>> L.maximal_order_infinite() #
↳ needs sage.rings.finite_rings
Maximal infinite order of Function field in y defined by  $y^2 + y + (x^2 + 1)/x$ 

```

monic_integral_model (names=None)

Return a function field isomorphic to this field but which is an extension of a rational function field with defining polynomial that is monic and integral over the constant base field.

INPUT:

- names – string or tuple of up to two strings (default: None); the name of the generator of the field, and the name of the generator of the underlying rational function field (if a tuple). If not given, then the names are chosen automatically.

OUTPUT:

A triple (F, f, t) where F is a function field, f is an isomorphism from F to this field, and t is the inverse of f .

EXAMPLES:

```

sage: K.<x> = FunctionField(QQ)
sage: R.<y> = K[]
sage: L.<y> = K.extension(x^2*y^5 - 1/x); L
Function field in y defined by  $x^2y^5 - 1/x$ 
sage: A, from_A, to_A = L.monic_integral_model('z')
sage: A
Function field in z defined by  $z^5 - x^{12}$ 
sage: from_A
Function Field morphism:
  From: Function field in z defined by  $z^5 - x^{12}$ 
  To:   Function field in y defined by  $x^2y^5 - 1/x$ 
  Defn:  $z \mapsto x^3y$ 

```

(continues on next page)

(continued from previous page)

```

      x |--> x
sage: to_A
Function Field morphism:
  From: Function field in y defined by  $x^2y^5 - 1/x$ 
  To:   Function field in z defined by  $z^5 - x^{12}$ 
  Defn:  $y \mapsto 1/x^3z$ 
      x |--> x
sage: to_A(y)
1/x^3*z
sage: from_A(to_A(y))
y
sage: from_A(to_A(1/y))
x^3*y^4
sage: from_A(to_A(1/y)) == 1/y
True

```

```

>>> from sage.all import *
>>> K = FunctionField(QQ, names=('x',)); (x,) = K._first_ngens(1)
>>> R = K['y']; (y,) = R._first_ngens(1)
>>> L = K.extension(x**Integer(2)*y**Integer(5) - Integer(1)/x, names=('y',));
↪ (y,) = L._first_ngens(1); L
Function field in y defined by  $x^2y^5 - 1/x$ 
>>> A, from_A, to_A = L.monic_integral_model('z')
>>> A
Function field in z defined by  $z^5 - x^{12}$ 
>>> from_A
Function Field morphism:
  From: Function field in z defined by  $z^5 - x^{12}$ 
  To:   Function field in y defined by  $x^2y^5 - 1/x$ 
  Defn:  $z \mapsto x^3y$ 
      x |--> x
>>> to_A
Function Field morphism:
  From: Function field in y defined by  $x^2y^5 - 1/x$ 
  To:   Function field in z defined by  $z^5 - x^{12}$ 
  Defn:  $y \mapsto 1/x^3z$ 
      x |--> x
>>> to_A(y)
1/x^3*z
>>> from_A(to_A(y))
y
>>> from_A(to_A(Integer(1)/y))
x^3*y^4
>>> from_A(to_A(Integer(1)/y)) == Integer(1)/y
True

```

This also works for towers of function fields:

```

sage: R.<z> = L[]
sage: M.<z> = L.extension(z^2*y - 1/x)
sage: M.monic_integral_model()
(Function field in z_ defined by  $z^{10} - x^{18}$ ,

```

(continues on next page)

(continued from previous page)

```

Function Field morphism:
  From: Function field in z_ defined by z_10 - x18
  To:   Function field in z defined by y*z2 - 1/x
  Defn: z_ |--> x2*z
         x |--> x, Function Field morphism:
  From: Function field in z defined by y*z2 - 1/x
  To:   Function field in z_ defined by z_10 - x18
  Defn: z |--> 1/x2*z_
         y |--> 1/x15*z_8
         x |--> x)

```

```

>>> from sage.all import *
>>> R = L['z']; (z,) = R._first_ngens(1)
>>> M = L.extension(z**Integer(2)*y - Integer(1)/x, names=('z',)); (z,) = M._
↪first_ngens(1)
>>> M.monic_integral_model()
(Function field in z_ defined by z_10 - x18,
Function Field morphism:
  From: Function field in z_ defined by z_10 - x18
  To:   Function field in z defined by y*z2 - 1/x
  Defn: z_ |--> x2*z
         x |--> x, Function Field morphism:
  From: Function field in z defined by y*z2 - 1/x
  To:   Function field in z_ defined by z_10 - x18
  Defn: z |--> 1/x2*z_
         y |--> 1/x15*z_8
         x |--> x)

```

ngens()

Return the number of generators of the function field over its base field. This is by definition 1.

EXAMPLES:

```

sage: K.<x> = FunctionField(QQ); R.<y> = K[]
sage: L.<y> = K.extension(y5 - (x3 + 2*x*y + 1/x))
sage: L.ngens()
1

```

```

>>> from sage.all import *
>>> K = FunctionField(QQ, names=('x',)); (x,) = K._first_ngens(1); R = K['y'];
↪ (y,) = R._first_ngens(1)
>>> L = K.extension(y**Integer(5) - (x**Integer(3) + Integer(2)*x*y +
↪ Integer(1)/x), names=('y',)); (y,) = L._first_ngens(1)
>>> L.ngens()
1

```

polynomial()

Return the univariate polynomial that defines the function field, that is, the polynomial $f(y)$ so that the function field is of the form $K[y]/(f(y))$.

EXAMPLES:

```

sage: K.<x> = FunctionField(QQ); R.<y> = K[]
sage: L.<y> = K.extension(y^5 - (x^3 + 2*x*y + 1/x))
sage: L.polynomial()
y^5 - 2*x*y + (-x^4 - 1)/x

```

```

>>> from sage.all import *
>>> K = FunctionField(QQ, names=('x',)); (x,) = K._first_ngens(1); R = K['y'];
↪ (y,) = R._first_ngens(1)
>>> L = K.extension(y**Integer(5) - (x**Integer(3) + Integer(2)*x*y +
↪ Integer(1)/x), names=('y',)); (y,) = L._first_ngens(1)
>>> L.polynomial()
y^5 - 2*x*y + (-x^4 - 1)/x

```

polynomial_ring()

Return the polynomial ring used to represent elements of the function field. If we view the function field as being presented as $K[y]/(f(y))$, then this function returns the ring $K[y]$.

EXAMPLES:

```

sage: K.<x> = FunctionField(QQ); R.<y> = K[]
sage: L.<y> = K.extension(y^5 - (x^3 + 2*x*y + 1/x))
sage: L.polynomial_ring()
Univariate Polynomial Ring in y over Rational function field in x over
↪ Rational Field

```

```

>>> from sage.all import *
>>> K = FunctionField(QQ, names=('x',)); (x,) = K._first_ngens(1); R = K['y'];
↪ (y,) = R._first_ngens(1)
>>> L = K.extension(y**Integer(5) - (x**Integer(3) + Integer(2)*x*y +
↪ Integer(1)/x), names=('y',)); (y,) = L._first_ngens(1)
>>> L.polynomial_ring()
Univariate Polynomial Ring in y over Rational function field in x over
↪ Rational Field

```

primitive_element()

Return a primitive element over the underlying rational function field.

If this is a finite extension of a rational function field $K(x)$ with K perfect, then this is a simple extension of $K(x)$, i.e., there is a primitive element y which generates this field over $K(x)$. This method returns such an element y .

EXAMPLES:

```

sage: K.<x> = FunctionField(QQ)
sage: R.<y> = K[]
sage: L.<y> = K.extension(y^2 - x)
sage: R.<z> = L[]
sage: M.<z> = L.extension(z^2 - y)
sage: R.<z> = L[]
sage: N.<u> = L.extension(z^2 - x - 1)
sage: N.primitive_element()
u + y
sage: M.primitive_element()
z

```

(continues on next page)

(continued from previous page)

```
sage: L.primitive_element()
y
```

```
>>> from sage.all import *
>>> K = FunctionField(QQ, names=('x',)); (x,) = K._first_ngens(1)
>>> R = K['y']; (y,) = R._first_ngens(1)
>>> L = K.extension(y**Integer(2) - x, names=('y',)); (y,) = L._first_ngens(1)
>>> R = L['z']; (z,) = R._first_ngens(1)
>>> M = L.extension(z**Integer(2) - y, names=('z',)); (z,) = M._first_ngens(1)
>>> R = L['z']; (z,) = R._first_ngens(1)
>>> N = L.extension(z**Integer(2) - x - Integer(1), names=('u',)); (u,) = N._
↳first_ngens(1)
>>> N.primitive_element()
u + y
>>> M.primitive_element()
z
>>> L.primitive_element()
y
```

This also works for inseparable extensions:

```
sage: # needs sage.rings.finite_rings
sage: K.<x> = FunctionField(GF(2))
sage: R.<Y> = K[]
sage: L.<y> = K.extension(Y^2 - x)
sage: R.<Z> = L[]
sage: M.<z> = L.extension(Z^2 - y)
sage: M.primitive_element()
z
```

```
>>> from sage.all import *
>>> # needs sage.rings.finite_rings
>>> K = FunctionField(GF(Integer(2)), names=('x',)); (x,) = K._first_ngens(1)
>>> R = K['Y']; (Y,) = R._first_ngens(1)
>>> L = K.extension(Y**Integer(2) - x, names=('y',)); (y,) = L._first_ngens(1)
>>> R = L['Z']; (Z,) = R._first_ngens(1)
>>> M = L.extension(Z**Integer(2) - y, names=('z',)); (z,) = M._first_ngens(1)
>>> M.primitive_element()
z
```

random_element (*args, **kwargs)

Create a random element of the function field. Parameters are passed onto the random_element method of the base_field.

EXAMPLES:

```
sage: K.<x> = FunctionField(QQ); R.<y> = K[]
sage: L.<y> = K.extension(y^2 - (x^2 + x))
sage: L.random_element() # random
((x^2 - x + 2/3)/(x^2 + 1/3*x - 1))*y^2 + ((-1/4*x^2 + 1/2*x - 1)/(-5/2*x + 2/
↳3))*y
+ (-1/2*x^2 - 4)/(-12*x^2 + 1/2*x - 1/95)
```

```

>>> from sage.all import *
>>> K = FunctionField(QQ, names=('x',)); (x,) = K._first_ngens(1); R = K['y'];
↪ (y,) = R._first_ngens(1)
>>> L = K.extension(y**Integer(2) - (x**Integer(2) + x), names=('y',)); (y,) ↪
↪ L._first_ngens(1)
>>> L.random_element() # random
((x^2 - x + 2/3)/(x^2 + 1/3*x - 1))*y^2 + ((-1/4*x^2 + 1/2*x - 1)/(-5/2*x + 2/
↪ 3))*y
+ (-1/2*x^2 - 4)/(-12*x^2 + 1/2*x - 1/95)

```

separable_model (*names=None*)

Return a function field isomorphic to this field which is a separable extension of a rational function field.

INPUT:

- *names* – tuple of two strings or *None* (default: *None*); the second entry will be used as the variable name of the rational function field, the first entry will be used as the variable name of its separable extension. If *None*, then the variable names will be chosen automatically.

OUTPUT:

A triple (F, f, t) where F is a function field, f is an isomorphism from F to this function field, and t is the inverse of f .

ALGORITHM:

Suppose that the constant base field is perfect. If this is a monic integral inseparable extension of a rational function field, then the defining polynomial is separable if we swap the variables (Proposition 4.8 in Chapter VIII of [Lan2002].) The algorithm reduces to this case with `monic_integral_model()`.

EXAMPLES:

```

sage: # needs sage.rings.finite_rings
sage: K.<x> = FunctionField(GF(2))
sage: R.<y> = K[]
sage: L.<y> = K.extension(y^2 - x^3)
sage: L.separable_model(('t', 'w'))
(Function field in t defined by t^3 + w^2,
Function Field morphism:
  From: Function field in t defined by t^3 + w^2
  To:   Function field in y defined by y^2 + x^3
  Defn: t |--> x
        w |--> y,
Function Field morphism:
  From: Function field in y defined by y^2 + x^3
  To:   Function field in t defined by t^3 + w^2
  Defn: y |--> w
        x |--> t)

```

```

>>> from sage.all import *
>>> # needs sage.rings.finite_rings
>>> K = FunctionField(GF(Integer(2)), names=('x',)); (x,) = K._first_ngens(1)
>>> R = K['y']; (y,) = R._first_ngens(1)
>>> L = K.extension(y**Integer(2) - x**Integer(3), names=('y',)); (y,) = L._
↪ first_ngens(1)
>>> L.separable_model(('t', 'w'))

```

(continues on next page)

(continued from previous page)

```
(Function field in t defined by  $t^3 + w^2$ ,
Function Field morphism:
  From: Function field in t defined by  $t^3 + w^2$ 
  To:   Function field in y defined by  $y^2 + x^3$ 
  Defn:  $t \mapsto x$ 
         $w \mapsto y$ ,
Function Field morphism:
  From: Function field in y defined by  $y^2 + x^3$ 
  To:   Function field in t defined by  $t^3 + w^2$ 
  Defn:  $y \mapsto w$ 
         $x \mapsto t$ )
```

This also works for non-integral polynomials:

```
sage: # needs sage.rings.finite_rings
sage: K.<x> = FunctionField(GF(2))
sage: R.<y> = K[]
sage: L.<y> = K.extension(y^2/x - x^2)
sage: L.separable_model()
(Function field in y_ defined by  $y_^3 + x_^2$ ,
Function Field morphism:
  From: Function field in y_ defined by  $y_^3 + x_^2$ 
  To:   Function field in y defined by  $1/x*y^2 + x^2$ 
  Defn:  $y_ \mapsto x$ 
         $x_ \mapsto y$ ,
Function Field morphism:
  From: Function field in y defined by  $1/x*y^2 + x^2$ 
  To:   Function field in y_ defined by  $y_^3 + x_^2$ 
  Defn:  $y \mapsto x_$ 
         $x \mapsto y_$ )
```

```
>>> from sage.all import *
>>> # needs sage.rings.finite_rings
>>> K = FunctionField(GF(Integer(2)), names=('x',)); (x,) = K._first_ngens(1)
>>> R = K['y']; (y,) = R._first_ngens(1)
>>> L = K.extension(y**Integer(2)/x - x**Integer(2), names=('y',)); (y,) = L._
↪first_ngens(1)
>>> L.separable_model()
(Function field in y_ defined by  $y_^3 + x_^2$ ,
Function Field morphism:
  From: Function field in y_ defined by  $y_^3 + x_^2$ 
  To:   Function field in y defined by  $1/x*y^2 + x^2$ 
  Defn:  $y_ \mapsto x$ 
         $x_ \mapsto y$ ,
Function Field morphism:
  From: Function field in y defined by  $1/x*y^2 + x^2$ 
  To:   Function field in y_ defined by  $y_^3 + x_^2$ 
  Defn:  $y \mapsto x_$ 
         $x \mapsto y_$ )
```

If the base field is not perfect this is only implemented in trivial cases:

```

sage: # needs sage.rings.finite_rings
sage: k.<t> = FunctionField(GF(2))
sage: k.is_perfect()
False
sage: K.<x> = FunctionField(k)
sage: R.<y> = K[]
sage: L.<y> = K.extension(y^3 - t)
sage: L.separable_model()
(Function field in y defined by y^3 + t,
 Function Field endomorphism of Function field in y defined by y^3 + t
  Defn: y |--> y
        x |--> x,
 Function Field endomorphism of Function field in y defined by y^3 + t
  Defn: y |--> y
        x |--> x)

```

```

>>> from sage.all import *
>>> # needs sage.rings.finite_rings
>>> k = FunctionField(GF(Integer(2)), names=('t',)); (t,) = k._first_ngens(1)
>>> k.is_perfect()
False
>>> K = FunctionField(k, names=('x',)); (x,) = K._first_ngens(1)
>>> R = K['y']; (y,) = R._first_ngens(1)
>>> L = K.extension(y**Integer(3) - t, names=('y',)); (y,) = L._first_ngens(1)
>>> L.separable_model()
(Function field in y defined by y^3 + t,
 Function Field endomorphism of Function field in y defined by y^3 + t
  Defn: y |--> y
        x |--> x,
 Function Field endomorphism of Function field in y defined by y^3 + t
  Defn: y |--> y
        x |--> x)

```

Some other cases for which a separable model could be constructed are not supported yet:

```

sage: R.<y> = K[] #_
↪needs sage.rings.finite_rings
sage: L.<y> = K.extension(y^2 - t) #_
↪needs sage.rings.finite_rings
sage: L.separable_model() #_
↪needs sage.rings.finite_rings
Traceback (most recent call last):
...
NotImplementedError: constructing a separable model is only implemented
for function fields over a perfect constant base field

```

```

>>> from sage.all import *
>>> R = K['y']; (y,) = R._first_ngens(1) # needs sage.rings.finite_rings
>>> L = K.extension(y**Integer(2) - t, names=('y',)); (y,) = L._first_ngens(1)
↪# needs sage.rings.finite_rings
>>> L.separable_model() #_
↪needs sage.rings.finite_rings

```

(continues on next page)

(continued from previous page)

```
Traceback (most recent call last):
...
NotImplementedError: constructing a separable model is only implemented
for function fields over a perfect constant base field
```

simple_model (*name=None*)

Return a function field isomorphic to this field which is a simple extension of a rational function field.

INPUT:

- *name* – string (default: None); the name of generator of the simple extension. If None, then the name of the generator will be the same as the name of the generator of this function field.

OUTPUT:

A triple (F, f, t) where F is a field isomorphic to this field, f is an isomorphism from F to this function field and t is the inverse of f .

EXAMPLES:

A tower of four function fields:

```
sage: K.<x> = FunctionField(QQ); R.<z> = K[]
sage: L.<z> = K.extension(z^2 - x); R.<u> = L[]
sage: M.<u> = L.extension(u^2 - z); R.<v> = M[]
sage: N.<v> = M.extension(v^2 - u)
```

```
>>> from sage.all import *
>>> K = FunctionField(QQ, names=('x',)); (x,) = K._first_ngens(1); R = K['z'];
↪ (z,) = R._first_ngens(1)
>>> L = K.extension(z**Integer(2) - x, names=('z',)); (z,) = L._first_
↪ ngens(1); R = L['u']; (u,) = R._first_ngens(1)
>>> M = L.extension(u**Integer(2) - z, names=('u',)); (u,) = M._first_
↪ ngens(1); R = M['v']; (v,) = R._first_ngens(1)
>>> N = M.extension(v**Integer(2) - u, names=('v',)); (v,) = N._first_ngens(1)
```

The fields N and M as simple extensions of K:

```
sage: N.simple_model()
(Function field in v defined by v^8 - x,
Function Field morphism:
  From: Function field in v defined by v^8 - x
  To:   Function field in v defined by v^2 - u
  Defn: v |--> v,
Function Field morphism:
  From: Function field in v defined by v^2 - u
  To:   Function field in v defined by v^8 - x
  Defn: v |--> v
        u |--> v^2
        z |--> v^4
        x |--> x)
sage: M.simple_model()
(Function field in u defined by u^4 - x,
Function Field morphism:
  From: Function field in u defined by u^4 - x
```

(continues on next page)

(continued from previous page)

```

To:   Function field in u defined by  $u^2 - z$ 
Defn:  $u \mapsto u$ ,
Function Field morphism:
From: Function field in u defined by  $u^2 - z$ 
To:   Function field in u defined by  $u^4 - x$ 
Defn:  $u \mapsto u$ 
       $z \mapsto u^2$ 
       $x \mapsto x$ )

```

```

>>> from sage.all import *
>>> N.simple_model()
(Function field in v defined by  $v^8 - x$ ,
Function Field morphism:
  From: Function field in v defined by  $v^8 - x$ 
  To:   Function field in v defined by  $v^2 - u$ 
  Defn:  $v \mapsto v$ ,
Function Field morphism:
  From: Function field in v defined by  $v^2 - u$ 
  To:   Function field in v defined by  $v^8 - x$ 
  Defn:  $v \mapsto v$ 
         $u \mapsto v^2$ 
         $z \mapsto v^4$ 
         $x \mapsto x$ )
>>> M.simple_model()
(Function field in u defined by  $u^4 - x$ ,
Function Field morphism:
  From: Function field in u defined by  $u^4 - x$ 
  To:   Function field in u defined by  $u^2 - z$ 
  Defn:  $u \mapsto u$ ,
Function Field morphism:
  From: Function field in u defined by  $u^2 - z$ 
  To:   Function field in u defined by  $u^4 - x$ 
  Defn:  $u \mapsto u$ 
         $z \mapsto u^2$ 
         $x \mapsto x$ )

```

An optional parameter `name` can be used to set the name of the generator of the simple extension:

```

sage: M.simple_model(name='t')
(Function field in t defined by  $t^4 - x$ , Function Field morphism:
  From: Function field in t defined by  $t^4 - x$ 
  To:   Function field in u defined by  $u^2 - z$ 
  Defn:  $t \mapsto u$ , Function Field morphism:
  From: Function field in u defined by  $u^2 - z$ 
  To:   Function field in t defined by  $t^4 - x$ 
  Defn:  $u \mapsto t$ 
         $z \mapsto t^2$ 
         $x \mapsto x$ )

```

```

>>> from sage.all import *
>>> M.simple_model(name='t')
(Function field in t defined by  $t^4 - x$ , Function Field morphism:

```

(continues on next page)

(continued from previous page)

```

From: Function field in t defined by  $t^4 - x$ 
To:   Function field in u defined by  $u^2 - z$ 
Defn:  $t \mapsto u$ , Function Field morphism:
From: Function field in u defined by  $u^2 - z$ 
To:   Function field in t defined by  $t^4 - x$ 
Defn:  $u \mapsto t$ 
       $z \mapsto t^2$ 
       $x \mapsto x$ 

```

An example with higher degrees:

```

sage: # needs sage.rings.finite_rings
sage: K.<x> = FunctionField(GF(3)); R.<y> = K[]
sage: L.<y> = K.extension(y^5 - x); R.<z> = L[]
sage: M.<z> = L.extension(z^3 - x)
sage: M.simple_model()
(Function field in z defined by  $z^{15} + xz^{12} + x^2z^9 + 2x^3z^6 + 2x^4z^3 + 2x^5 + 2x^3$ ,
↪ Function Field morphism:
  From: Function field in z defined by  $z^{15} + xz^{12} + x^2z^9 + 2x^3z^6 + 2x^4z^3 + 2x^5 + 2x^3$ ,
↪ To:   Function field in z defined by  $z^3 + 2x$ 
  Defn:  $z \mapsto z + y$ ,
Function Field morphism:
  From: Function field in z defined by  $z^3 + 2x$ 
  To:   Function field in z defined by  $z^{15} + xz^{12} + x^2z^9 + 2x^3z^6 + 2x^4z^3 + 2x^5 + 2x^3$ ,
↪ Defn:  $z \mapsto 2/xz^6 + 2z^3 + z + 2x$ 
       $y \mapsto 1/xz^6 + z^3 + x$ 
       $x \mapsto x$ )

```

```

>>> from sage.all import *
>>> # needs sage.rings.finite_rings
>>> K = FunctionField(GF(Integer(3)), names=('x',)); (x,) = K._first_ngens(1);
↪ R = K['y']; (y,) = R._first_ngens(1)
>>> L = K.extension(y**Integer(5) - x, names=('y',)); (y,) = L._first_
↪ ngens(1); R = L['z']; (z,) = R._first_ngens(1)
>>> M = L.extension(z**Integer(3) - x, names=('z',)); (z,) = M._first_ngens(1)
>>> M.simple_model()
(Function field in z defined by  $z^{15} + xz^{12} + x^2z^9 + 2x^3z^6 + 2x^4z^3 + 2x^5 + 2x^3$ ,
↪ Function Field morphism:
  From: Function field in z defined by  $z^{15} + xz^{12} + x^2z^9 + 2x^3z^6 + 2x^4z^3 + 2x^5 + 2x^3$ ,
↪ To:   Function field in z defined by  $z^3 + 2x$ 
  Defn:  $z \mapsto z + y$ ,
Function Field morphism:
  From: Function field in z defined by  $z^3 + 2x$ 
  To:   Function field in z defined by  $z^{15} + xz^{12} + x^2z^9 + 2x^3z^6 + 2x^4z^3 + 2x^5 + 2x^3$ ,
↪ Defn:  $z \mapsto 2/xz^6 + 2z^3 + z + 2x$ 
       $y \mapsto 1/xz^6 + z^3 + x$ 

```

(continues on next page)

(continued from previous page)

```
x |--> x)
```

This also works for inseparable extensions:

```
sage: # needs sage.rings.finite_rings
sage: K.<x> = FunctionField(GF(2)); R.<y> = K[]
sage: L.<y> = K.extension(y^2 - x); R.<z> = L[]
sage: M.<z> = L.extension(z^2 - y)
sage: M.simple_model()
(Function field in z defined by z^4 + x,
Function Field morphism:
  From: Function field in z defined by z^4 + x
  To:   Function field in z defined by z^2 + y
  Defn: z |--> z,
Function Field morphism:
  From: Function field in z defined by z^2 + y
  To:   Function field in z defined by z^4 + x
  Defn: z |--> z
        y |--> z^2
        x |--> x)
```

```
>>> from sage.all import *
>>> # needs sage.rings.finite_rings
>>> K = FunctionField(GF(Integer(2)), names=('x',)); (x,) = K._first_ngens(1);
↪ R = K['y']; (y,) = R._first_ngens(1)
>>> L = K.extension(y**Integer(2) - x, names=('y',)); (y,) = L._first_
↪ ngens(1); R = L['z']; (z,) = R._first_ngens(1)
>>> M = L.extension(z**Integer(2) - y, names=('z',)); (z,) = M._first_ngens(1)
>>> M.simple_model()
(Function field in z defined by z^4 + x,
Function Field morphism:
  From: Function field in z defined by z^4 + x
  To:   Function field in z defined by z^2 + y
  Defn: z |--> z,
Function Field morphism:
  From: Function field in z defined by z^2 + y
  To:   Function field in z defined by z^4 + x
  Defn: z |--> z
        y |--> z^2
        x |--> x)
```

```
class sage.rings.function_field.function_field_polymod.FunctionField_simple(polynomial,
                                                                              names, cate-
                                                                              gory=None)
```

Bases: *FunctionField_polymod*

Function fields defined by irreducible and separable polynomials over rational function fields.

constant_field()

Return the algebraic closure of the base constant field in the function field.

EXAMPLES:

```

sage: K.<x> = FunctionField(GF(3)); _.<y> = K[] #_
↪needs sage.rings.finite_rings
sage: L.<y> = K.extension(y^5 - (x^3 + 2*x*y + 1/x)) #_
↪needs sage.rings.finite_rings
sage: L.constant_field() #_
↪needs sage.rings.finite_rings
Finite Field of size 3

```

```

>>> from sage.all import *
>>> K = FunctionField(GF(Integer(3)), names=('x',)); (x,) = K._first_ngens(1);
↪ _ = K['y']; (y,) = _.first_ngens(1) # needs sage.rings.finite_rings
>>> L = K.extension(y**Integer(5) - (x**Integer(3) + Integer(2)*x*y +
↪ Integer(1)/x), names=('y',)); (y,) = L._first_ngens(1) # needs sage.rings.
↪ finite_rings
>>> L.constant_field() #_
↪needs sage.rings.finite_rings
Finite Field of size 3

```

exact_constant_field(name='t')

Return the exact constant field and its embedding into the function field.

INPUT:

- name – name (default: *t*) of the generator of the exact constant field

EXAMPLES:

```

sage: # needs sage.rings.finite_rings
sage: K.<x> = FunctionField(GF(3)); _.<Y> = K[]
sage: f = Y^2 - x*Y + x^2 + 1 # irreducible but not absolutely irreducible
sage: L.<y> = K.extension(f)
sage: L.genus()
0
sage: L.exact_constant_field()
(Finite Field in t of size 3^2, Ring morphism:
  From: Finite Field in t of size 3^2
  To:   Function field in y defined by y^2 + 2*x*y + x^2 + 1
  Defn: t |--> y + x)
sage: (y+x).divisor()
0

```

```

>>> from sage.all import *
>>> # needs sage.rings.finite_rings
>>> K = FunctionField(GF(Integer(3)), names=('x',)); (x,) = K._first_ngens(1);
↪ _ = K['Y']; (Y,) = _.first_ngens(1)
>>> f = Y**Integer(2) - x*Y + x**Integer(2) + Integer(1) # irreducible but_
↪ not absolutely irreducible
>>> L = K.extension(f, names=('y',)); (y,) = L._first_ngens(1)
>>> L.genus()
0
>>> L.exact_constant_field()
(Finite Field in t of size 3^2, Ring morphism:
  From: Finite Field in t of size 3^2
  To:   Function field in y defined by y^2 + 2*x*y + x^2 + 1

```

(continues on next page)

(continued from previous page)

```

    Defn: t |--> y + x)
>>> (y+x).divisor()
0

```

genus()

Return the genus of the function field.

EXAMPLES:

```

sage: # needs sage.rings.finite_rings
sage: F.<a> = GF(16)
sage: K.<x> = FunctionField(F); K
Rational function field in x over Finite Field in a of size 2^4
sage: R.<t> = PolynomialRing(K)
sage: L.<y> = K.extension(t^4 + t - x^5)
sage: L.genus()
6

sage: # needs sage.rings.number_field
sage: R.<T> = QQ[]
sage: N.<a> = NumberField(T^2 + 1)
sage: K.<x> = FunctionField(N); K
Rational function field in x over Number Field in a with defining polynomial
↳ T^2 + 1
sage: K.genus()
0
sage: S.<t> = PolynomialRing(K)
sage: L.<y> = K.extension(t^2 - x^3 + x)
sage: L.genus()
1

```

```

>>> from sage.all import *
>>> # needs sage.rings.finite_rings
>>> F = GF(Integer(16), names=('a',)); (a,) = F._first_ngens(1)
>>> K = FunctionField(F, names=('x',)); (x,) = K._first_ngens(1); K
Rational function field in x over Finite Field in a of size 2^4
>>> R = PolynomialRing(K, names=('t',)); (t,) = R._first_ngens(1)
>>> L = K.extension(t**Integer(4) + t - x**Integer(5), names=('y',)); (y,) =
↳ L._first_ngens(1)
>>> L.genus()
6

>>> # needs sage.rings.number_field
>>> R = QQ['T']; (T,) = R._first_ngens(1)
>>> N = NumberField(T**Integer(2) + Integer(1), names=('a',)); (a,) = N._
↳ first_ngens(1)
>>> K = FunctionField(N, names=('x',)); (x,) = K._first_ngens(1); K
Rational function field in x over Number Field in a with defining polynomial
↳ T^2 + 1
>>> K.genus()
0
>>> S = PolynomialRing(K, names=('t',)); (t,) = S._first_ngens(1)

```

(continues on next page)

(continued from previous page)

```

>>> L = K.extension(t**Integer(2) - x**Integer(3) + x, names=('y',)); (y,) = L
↳ L._first_ngens(1)
>>> L.genus()
1

```

The genus is computed by the Hurwitz genus formula.

`places_above(p)`

Return places lying above p .

INPUT:

- p – place of the base rational function field

EXAMPLES:

```

sage: K.<x> = FunctionField(GF(2)); _.<Y> = K[] #_
↳ needs sage.rings.finite_rings
sage: F.<y> = K.extension(Y^3 - x^2*(x^2 + x + 1)^2) #_
↳ needs sage.rings.finite_rings
sage: all(q.place_below() == p #_
↳ needs sage.rings.finite_rings
.....:     for p in K.places() for q in F.places_above(p))
True

sage: K.<x> = FunctionField(QQ); _.<Y> = K[]
sage: F.<y> = K.extension(Y^3 - x^2*(x^2 + x + 1)^2)
sage: O = K.maximal_order()
sage: pls = [O.ideal(x - c).place() for c in [-2, -1, 0, 1, 2]]
sage: all(q.place_below() == p
.....:     for p in pls for q in F.places_above(p))
True

sage: # needs sage.rings.number_field
sage: K.<x> = FunctionField(QQbar); _.<Y> = K[]
sage: F.<y> = K.extension(Y^3 - x^2*(x^2 + x + 1)^2)
sage: O = K.maximal_order()
sage: pls = [O.ideal(x - QQbar(sqrt(c))).place()
.....:     for c in [-2, -1, 0, 1, 2]]
sage: all(q.place_below() == p # long time (4s)
.....:     for p in pls for q in F.places_above(p))
True

```

```

>>> from sage.all import *
>>> K = FunctionField(GF(Integer(2)), names=('x',)); (x,) = K._first_ngens(1);
↳ _ = K['Y']; (Y,) = _.first_ngens(1) # needs sage.rings.finite_rings
>>> F = K.extension(Y**Integer(3) - x**Integer(2)*(x**Integer(2) + x +
↳ Integer(1))**Integer(2), names=('y',)); (y,) = F._first_ngens(1) # needs
↳ sage.rings.finite_rings
>>> all(q.place_below() == p #_
↳ needs sage.rings.finite_rings
...     for p in K.places() for q in F.places_above(p))
True

```

(continues on next page)

(continued from previous page)

```

>>> K = FunctionField(QQ, names=('x',)); (x,) = K._first_ngens(1); _ = K['Y'];
↳ (Y,) = _._first_ngens(1)
>>> F = K.extension(Y**Integer(3) - x**Integer(2)*(x**Integer(2) + x +
↳ Integer(1))**Integer(2), names=('y',)); (y,) = F._first_ngens(1)
>>> O = K.maximal_order()
>>> pls = [O.ideal(x - c).place() for c in [-Integer(2), -Integer(1),
↳ Integer(0), Integer(1), Integer(2)]]
>>> all(q.place_below() == p
...     for p in pls for q in F.places_above(p))
True

>>> # needs sage.rings.number_field
>>> K = FunctionField(QQbar, names=('x',)); (x,) = K._first_ngens(1); _ = K['Y']
↳ ; (Y,) = _._first_ngens(1)
>>> F = K.extension(Y**Integer(3) - x**Integer(2)*(x**Integer(2) + x +
↳ Integer(1))**Integer(2), names=('y',)); (y,) = F._first_ngens(1)
>>> O = K.maximal_order()
>>> pls = [O.ideal(x - QQbar(sqrt(c))).place()
...     for c in [-Integer(2), -Integer(1), Integer(0), Integer(1),
↳ Integer(2)]]
>>> all(q.place_below() == p # long time (4s)
...     for p in pls for q in F.places_above(p))
True

```

places_infinite(degree=1)

Return a list of the infinite places with degree.

INPUT:

- degree – positive integer (default: 1)

EXAMPLES:

```

sage: # needs sage.rings.finite_rings
sage: F.<a> = GF(2)
sage: K.<x> = FunctionField(F)
sage: R.<t> = PolynomialRing(K)
sage: L.<y> = K.extension(t^4 + t - x^5)
sage: L.places_infinite(1)
[Place (1/x, 1/x^4*y^3)]

```

```

>>> from sage.all import *
>>> # needs sage.rings.finite_rings
>>> F = GF(Integer(2), names=('a',)); (a,) = F._first_ngens(1)
>>> K = FunctionField(F, names=('x',)); (x,) = K._first_ngens(1)
>>> R = PolynomialRing(K, names=('t',)); (t,) = R._first_ngens(1)
>>> L = K.extension(t**Integer(4) + t - x**Integer(5), names=('y',)); (y,) =
↳ L._first_ngens(1)
>>> L.places_infinite(Integer(1))
[Place (1/x, 1/x^4*y^3)]

```

residue_field(place, name=None)

Return the residue field associated with the place along with the maps from and to the residue field.

INPUT:

- `place` – place of the function field
- `name` – string; name of the generator of the residue field

The domain of the map to the residue field is the discrete valuation ring associated with the place.

The discrete valuation ring is defined as the ring of all elements of the function field with nonnegative valuation at the place. The maximal ideal is the set of elements of positive valuation. The residue field is then the quotient of the discrete valuation ring by its maximal ideal.

If an element not in the valuation ring is applied to the map, an exception `TypeError` is raised.

EXAMPLES:

```
sage: # needs sage.rings.finite_rings
sage: K.<x> = FunctionField(GF(2)); _.<Y> = K[]
sage: L.<y> = K.extension(Y^2 + Y + x + 1/x)
sage: p = L.places_finite()[0]
sage: R, fr_R, to_R = L.residue_field(p)
sage: R
Finite Field of size 2
sage: f = 1 + y
sage: f.valuation(p)
-1
sage: to_R(f)
Traceback (most recent call last):
...
TypeError: ...
sage: (1+1/f).valuation(p)
0
sage: to_R(1 + 1/f)
1
sage: [fr_R(e) for e in R]
[0, 1]
```

```
>>> from sage.all import *
>>> # needs sage.rings.finite_rings
>>> K = FunctionField(GF(Integer(2)), names=('x',)); (x,) = K._first_ngens(1);
↪ _ = K['Y']; (Y,) = _._first_ngens(1)
>>> L = K.extension(Y**Integer(2) + Y + x + Integer(1)/x, names=('y',)); (y,) =
↪ L._first_ngens(1)
>>> p = L.places_finite()[Integer(0)]
>>> R, fr_R, to_R = L.residue_field(p)
>>> R
Finite Field of size 2
>>> f = Integer(1) + y
>>> f.valuation(p)
-1
>>> to_R(f)
Traceback (most recent call last):
...
TypeError: ...
>>> (Integer(1)+Integer(1)/f).valuation(p)
0
```

(continues on next page)

(continued from previous page)

```
>>> to_R(Integer(1) + Integer(1)/f)
1
>>> [fr_R(e) for e in R]
[0, 1]
```

ELEMENTS OF FUNCTION FIELDS

Sage provides arithmetic with elements of function fields.

EXAMPLES:

Arithmetic with rational functions:

```
sage: K.<t> = FunctionField(QQ)
sage: f = t - 1
sage: g = t^2 - 3
sage: h = f^2/g^3
sage: h.valuation(t-1)
2
sage: h.valuation(t)
0
sage: h.valuation(t^2 - 3)
-3
```

```
>>> from sage.all import *
>>> K = FunctionField(QQ, names=('t',)); (t,) = K._first_ngens(1)
>>> f = t - Integer(1)
>>> g = t**Integer(2) - Integer(3)
>>> h = f**Integer(2)/g**Integer(3)
>>> h.valuation(t-Integer(1))
2
>>> h.valuation(t)
0
>>> h.valuation(t**Integer(2) - Integer(3))
-3
```

Derivatives of elements in separable extensions:

```
sage: K.<x> = FunctionField(GF(4)); _.<Y> = K[] #_
↳needs sage.rings.finite_rings
sage: L.<y> = K.extension(Y^2 + Y + x + 1/x) #_
↳needs sage.rings.finite_rings sage.rings.function_field
sage: (y^3 + x).derivative() #_
↳needs sage.rings.finite_rings sage.rings.function_field
((x^2 + 1)/x^2)*y + (x^4 + x^3 + 1)/x^3
```

```
>>> from sage.all import *
>>> K = FunctionField(GF(Integer(4)), names=('x',)); (x,) = K._first_ngens(1); _ = K[
(continues on next page)
```

(continued from previous page)

```

↪ 'Y']; (Y,) = _._first_ngens(1) # needs sage.rings.finite_rings
>>> L = K.extension(Y**Integer(2) + Y + x + Integer(1)/x, names=('y',)); (y,) = L._
↪ first_ngens(1) # needs sage.rings.finite_rings sage.rings.function_field
>>> (y**Integer(3) + x).derivative()
↪ # needs sage.rings.finite_rings sage.rings.function_field
((x^2 + 1)/x^2)*y + (x^4 + x^3 + 1)/x^3

```

The divisor of an element of a global function field:

```

sage: K.<x> = FunctionField(GF(2)); _.<Y> = K[]
sage: L.<y> = K.extension(Y^2 + Y + x + 1/x) #_
↪ needs sage.rings.function_field
sage: y.divisor() #_
↪ needs sage.rings.function_field
- Place (1/x, 1/x*y)
- Place (x, x*y)
+ 2*Place (x + 1, x*y)

```

```

>>> from sage.all import *
>>> K = FunctionField(GF(Integer(2)), names=('x',)); (x,) = K._first_ngens(1); _ = K[
↪ 'Y']; (Y,) = _._first_ngens(1)
>>> L = K.extension(Y**Integer(2) + Y + x + Integer(1)/x, names=('y',)); (y,) = L._
↪ first_ngens(1) # needs sage.rings.function_field
>>> y.divisor() #_
↪ needs sage.rings.function_field
- Place (1/x, 1/x*y)
- Place (x, x*y)
+ 2*Place (x + 1, x*y)

```

AUTHORS:

- William Stein: initial version
- Robert Bradshaw (2010-05-27): cythonize function field elements
- Julian Rueth (2011-06-28, 2020-09-01): treat zero correctly; implement `nth_root/is_nth_power`
- Maarten Derickx (2011-09-11): added doctests, fixed pickling
- Kwankyu Lee (2017-04-30): added elements for global function fields
- Vincent Macri (2024-09-03): added subs method

class sage.rings.function_field.element.**FunctionFieldElement**

Bases: `FieldElement`

Abstract base class for function field elements.

EXAMPLES:

```

sage: t = FunctionField(QQ, 't').gen()
sage: isinstance(t, sage.rings.function_field.element.FunctionFieldElement)
True

```

```

>>> from sage.all import *
>>> t = FunctionField(QQ, 't').gen()

```

(continues on next page)

(continued from previous page)

```
>>> isinstance(t, sage.rings.function_field.element.FunctionFieldElement)
True
```

characteristic_polynomial (*args, **kws)

Return the characteristic polynomial of the element. Give an optional input string to name the variable in the characteristic polynomial.

EXAMPLES:

```
sage: K.<x> = FunctionField(QQ); R.<y> = K[]
sage: x.characteristic_polynomial('W') #_
↳needs sage.modules
W - x

sage: # needs sage.rings.function_field
sage: L.<y> = K.extension(y^2 - x*y + 4*x^3); R.<z> = L[]
sage: M.<z> = L.extension(z^3 - y^2*z + x)
sage: y.characteristic_polynomial('W')
W^2 - x*W + 4*x^3
sage: z.characteristic_polynomial('W')
W^3 + (-x*y + 4*x^3)*W + x
```

```
>>> from sage.all import *
>>> K = FunctionField(QQ, names=('x',)); (x,) = K._first_ngens(1); R = K['y'];
↳ (y,) = R._first_ngens(1)
>>> x.characteristic_polynomial('W') #_
↳needs sage.modules
W - x

>>> # needs sage.rings.function_field
>>> L = K.extension(y**Integer(2) - x*y + Integer(4)*x**Integer(3), names=('y',)); (y,) = L._first_ngens(1); R = L['z']; (z,) = R._first_ngens(1)
↳ (z,) = R._first_ngens(1)
>>> M = L.extension(z**Integer(3) - y**Integer(2)*z + x, names=('z',)); (z,) = M._first_ngens(1)
↳ (z,) = M._first_ngens(1)
>>> y.characteristic_polynomial('W')
W^2 - x*W + 4*x^3
>>> z.characteristic_polynomial('W')
W^3 + (-x*y + 4*x^3)*W + x
```

charpoly (*args, **kws)

Return the characteristic polynomial of the element. Give an optional input string to name the variable in the characteristic polynomial.

EXAMPLES:

```
sage: K.<x> = FunctionField(QQ); R.<y> = K[]
sage: x.characteristic_polynomial('W') #_
↳needs sage.modules
W - x

sage: # needs sage.rings.function_field
sage: L.<y> = K.extension(y^2 - x*y + 4*x^3); R.<z> = L[]
sage: M.<z> = L.extension(z^3 - y^2*z + x)
```

(continues on next page)

(continued from previous page)

```
sage: y.characteristic_polynomial('W')
W^2 - x*W + 4*x^3
sage: z.characteristic_polynomial('W')
W^3 + (-x*y + 4*x^3)*W + x
```

```
>>> from sage.all import *
>>> K = FunctionField(QQ, names=('x',)); (x,) = K._first_ngens(1); R = K['y'];
↪ (y,) = R._first_ngens(1)
>>> x.characteristic_polynomial('W')                                     #_
↪ needs sage.modules
W - x

>>> # needs sage.rings.function_field
>>> L = K.extension(y**Integer(2) - x*y + Integer(4)*x**Integer(3), names=('y
↪ ',)); (y,) = L._first_ngens(1); R = L['z']; (z,) = R._first_ngens(1)
>>> M = L.extension(z**Integer(3) - y**Integer(2)*z + x, names=('z',)); (z,) ↪
↪ M._first_ngens(1)
>>> y.characteristic_polynomial('W')
W^2 - x*W + 4*x^3
>>> z.characteristic_polynomial('W')
W^3 + (-x*y + 4*x^3)*W + x
```

degree()

Return the max degree between the denominator and numerator.

EXAMPLES:

```
sage: FF.<t> = FunctionField(QQ)
sage: f = (t^2 + 3) / (t^3 - 1/3); f
(t^2 + 3)/(t^3 - 1/3)
sage: f.degree()
3

sage: FF.<t> = FunctionField(QQ)
sage: f = (t+8); f
t + 8
sage: f.degree()
1
```

```
>>> from sage.all import *
>>> FF = FunctionField(QQ, names=('t',)); (t,) = FF._first_ngens(1)
>>> f = (t**Integer(2) + Integer(3)) / (t**Integer(3) - Integer(1)/
↪ Integer(3)); f
(t^2 + 3)/(t^3 - 1/3)
>>> f.degree()
3

>>> FF = FunctionField(QQ, names=('t',)); (t,) = FF._first_ngens(1)
>>> f = (t+Integer(8)); f
t + 8
>>> f.degree()
1
```


derivative()

Return the derivative of the element.

The derivative is with respect to the generator of the base rational function field, over which the function field is a separable extension.

EXAMPLES:

```
sage: K.<t> = FunctionField(QQ)
sage: f = (t + 1) / (t^2 - 1/3)
sage: f.derivative()                                     #_
↳needs sage.modules
(-t^2 - 2*t - 1/3)/(t^4 - 2/3*t^2 + 1/9)

sage: K.<x> = FunctionField(GF(4)); _.<Y> = K[]          #_
↳needs sage.rings.finite_rings
sage: L.<y> = K.extension(Y^2 + Y + x + 1/x)            #_
↳needs sage.rings.finite_rings sage.rings.function_field
sage: (y^3 + x).derivative()                           #_
↳needs sage.rings.finite_rings sage.rings.function_field
((x^2 + 1)/x^2)*y + (x^4 + x^3 + 1)/x^3

>>> from sage.all import *
>>> K = FunctionField(QQ, names=('t',)); (t,) = K._first_ngens(1)
>>> f = (t + Integer(1)) / (t**Integer(2) - Integer(1)/Integer(3))
>>> f.derivative()                                     #_
↳needs sage.modules
(-t^2 - 2*t - 1/3)/(t^4 - 2/3*t^2 + 1/9)

>>> K = FunctionField(GF(Integer(4)), names=('x',)); (x,) = K._first_ngens(1);
↳ _ = K['Y']; (Y,) = _.first_ngens(1) # needs sage.rings.finite_rings
>>> L = K.extension(Y**Integer(2) + Y + x + Integer(1)/x, names=('y',)); (y,) =
↳ L._first_ngens(1) # needs sage.rings.finite_rings sage.rings.function_field
>>> (y**Integer(3) + x).derivative()                   #_
↳ # needs sage.rings.finite_rings sage.rings.function_field
((x^2 + 1)/x^2)*y + (x^4 + x^3 + 1)/x^3
```

differential()

Return the differential dx where x is the element.

EXAMPLES:

```
sage: K.<t> = FunctionField(QQ)
sage: f = 1 / t
sage: f.differential()                                 #_
↳needs sage.modules
(-1/t^2) d(t)

sage: K.<x> = FunctionField(GF(4)); _.<Y> = K[]          #_
↳needs sage.rings.finite_rings
sage: L.<y> = K.extension(Y^2 + Y + x + 1/x)            #_
↳needs sage.rings.finite_rings sage.rings.function_field
sage: (y^3 + x).differential()                         #_
↳needs sage.rings.finite_rings sage.rings.function_field
(((x^2 + 1)/x^2)*y + (x^4 + x^3 + 1)/x^3) d(x)
```

```

>>> from sage.all import *
>>> K = FunctionField(QQ, names=('t',)); (t,) = K._first_ngens(1)
>>> f = Integer(1) / t
>>> f.differential()
↳needs sage.modules
(-1/t^2) d(t)

>>> K = FunctionField(GF(Integer(4)), names=('x',)); (x,) = K._first_ngens(1);
↳ _ = K['Y']; (Y,) = _._first_ngens(1) # needs sage.rings.finite_rings
>>> L = K.extension(Y**Integer(2) + Y + x + Integer(1)/x, names=('y',)); (y,)
↳ = L._first_ngens(1) # needs sage.rings.finite_rings sage.rings.function_field
>>> (y**Integer(3) + x).differential()
↳ # needs sage.rings.finite_rings sage.rings.function_field
((x^2 + 1)/x^2)*y + (x^4 + x^3 + 1)/x^3 d(x)

```

divisor()

Return the divisor of the element.

EXAMPLES:

```

sage: K.<x> = FunctionField(GF(2))
sage: f = 1/(x^3 + x^2 + x)
sage: f.divisor()
↳needs sage.libs.pari sage.modules
3*Place (1/x)
- Place (x)
- Place (x^2 + x + 1)

```

```

>>> from sage.all import *
>>> K = FunctionField(GF(Integer(2)), names=('x',)); (x,) = K._first_ngens(1)
>>> f = Integer(1)/(x**Integer(3) + x**Integer(2) + x)
>>> f.divisor()
↳needs sage.libs.pari sage.modules
3*Place (1/x)
- Place (x)
- Place (x^2 + x + 1)

```

```

sage: K.<x> = FunctionField(GF(2)); _.<Y> = K[]
sage: L.<y> = K.extension(Y^2 + Y + x + 1/x)
↳needs sage.rings.function_field
sage: y.divisor()
↳needs sage.rings.function_field
- Place (1/x, 1/x*y)
- Place (x, x*y)
+ 2*Place (x + 1, x*y)

```

```

>>> from sage.all import *
>>> K = FunctionField(GF(Integer(2)), names=('x',)); (x,) = K._first_ngens(1);
↳ _ = K['Y']; (Y,) = _._first_ngens(1)
>>> L = K.extension(Y**Integer(2) + Y + x + Integer(1)/x, names=('y',)); (y,)
↳ = L._first_ngens(1) # needs sage.rings.function_field
>>> y.divisor()
↳

```

(continues on next page)

(continued from previous page)

```

↪needs sage.rings.function_field
- Place (1/x, 1/x*y)
- Place (x, x*y)
+ 2*Place (x + 1, x*y)

```

divisor_of_poles()

Return the divisor of poles for the element.

EXAMPLES:

```

sage: K.<x> = FunctionField(GF(2))
sage: f = 1/(x^3 + x^2 + x)
sage: f.divisor_of_poles() #_
↪needs sage.libs.pari sage.modules
Place (x)
+ Place (x^2 + x + 1)

```

```

>>> from sage.all import *
>>> K = FunctionField(GF(Integer(2)), names=('x',)); (x,) = K._first_ngens(1)
>>> f = Integer(1)/(x**Integer(3) + x**Integer(2) + x)
>>> f.divisor_of_poles() #_
↪needs sage.libs.pari sage.modules
Place (x)
+ Place (x^2 + x + 1)

```

```

sage: K.<x> = FunctionField(GF(4)); _.<Y> = K[] #_
↪needs sage.rings.finite_rings
sage: L.<y> = K.extension(Y^2 + Y + x + 1/x) #_
↪needs sage.rings.finite_rings sage.rings.function_field
sage: (x/y).divisor_of_poles() #_
↪needs sage.rings.finite_rings sage.rings.function_field
Place (1/x, 1/x*y) + 2*Place (x + 1, x*y)

```

```

>>> from sage.all import *
>>> K = FunctionField(GF(Integer(4)), names=('x',)); (x,) = K._first_ngens(1);
↪ _ = K['Y']; (Y,) = _.first_ngens(1) # needs sage.rings.finite_rings
>>> L = K.extension(Y**Integer(2) + Y + x + Integer(1)/x, names=('y',)); (y,) =
↪ L._first_ngens(1) # needs sage.rings.finite_rings sage.rings.function_field
>>> (x/y).divisor_of_poles() #_
↪needs sage.rings.finite_rings sage.rings.function_field
Place (1/x, 1/x*y) + 2*Place (x + 1, x*y)

```

divisor_of_zeros()

Return the divisor of zeros for the element.

EXAMPLES:

```

sage: K.<x> = FunctionField(GF(2))
sage: f = 1/(x^3 + x^2 + x)
sage: f.divisor_of_zeros() #_
↪needs sage.libs.pari sage.modules
3*Place (1/x)

```

```
>>> from sage.all import *
>>> K = FunctionField(GF(Integer(2)), names=('x',)); (x,) = K._first_ngens(1)
>>> f = Integer(1)/(x**Integer(3) + x**Integer(2) + x)
>>> f.divisor_of_zeros() #_
↳needs sage.libs.pari sage.modules
3*Place (1/x)
```

```
sage: K.<x> = FunctionField(GF(4)); _.<Y> = K[] #_
↳needs sage.rings.finite_rings
sage: L.<y> = K.extension(Y^2 + Y + x + 1/x) #_
↳needs sage.rings.finite_rings sage.rings.function_field
sage: (x/y).divisor_of_zeros() #_
↳needs sage.rings.finite_rings sage.rings.function_field
3*Place (x, x*y)
```

```
>>> from sage.all import *
>>> K = FunctionField(GF(Integer(4)), names=('x',)); (x,) = K._first_ngens(1);
↳ _ = K['Y']; (Y,) = _.first_ngens(1) # needs sage.rings.finite_rings
>>> L = K.extension(Y**Integer(2) + Y + x + Integer(1)/x, names=('y',)); (y,)
↳ = L._first_ngens(1) # needs sage.rings.finite_rings sage.rings.function_field
>>> (x/y).divisor_of_zeros() #_
↳needs sage.rings.finite_rings sage.rings.function_field
3*Place (x, x*y)
```

evaluate (place)

Return the value of the element at the place.

INPUT:

- place – a function field place

OUTPUT:

If the element is in the valuation ring at the place, then an element in the residue field at the place is returned. Otherwise, a `ValueError` is raised.

EXAMPLES:

```
sage: K.<t> = FunctionField(GF(5))
sage: p = K.place_infinite()
sage: f = 1/t^2 + 3
sage: f.evaluate(p)
3
```

```
>>> from sage.all import *
>>> K = FunctionField(GF(Integer(5)), names=('t',)); (t,) = K._first_ngens(1)
>>> p = K.place_infinite()
>>> f = Integer(1)/t**Integer(2) + Integer(3)
>>> f.evaluate(p)
3
```

```
sage: # needs sage.rings.finite_rings sage.rings.function_field
sage: K.<x> = FunctionField(GF(4)); _.<Y> = K[]
sage: L.<y> = K.extension(Y^2 + Y + x + 1/x)
```

(continues on next page)

(continued from previous page)

```

sage: p, = L.places_infinite()
sage: p, = L.places_infinite()
sage: (y + x).evaluate(p)
Traceback (most recent call last):
...
ValueError: has a pole at the place
sage: (y/x + 1).evaluate(p)
1

```

```

>>> from sage.all import *
>>> # needs sage.rings.finite_rings sage.rings.function_field
>>> K = FunctionField(GF(Integer(4)), names=('x',)); (x,) = K._first_ngens(1);
↪ _ = K['Y']; (Y,) = _._first_ngens(1)
>>> L = K.extension(Y**Integer(2) + Y + x + Integer(1)/x, names=('y',)); (y,) ↪
↪ = L._first_ngens(1)
>>> p, = L.places_infinite()
>>> p, = L.places_infinite()
>>> (y + x).evaluate(p)
Traceback (most recent call last):
...
ValueError: has a pole at the place
>>> (y/x + Integer(1)).evaluate(p)
1

```

higher_derivative (*i*, *separating_element*=None)

Return the *i*-th derivative of the element with respect to the separating element.

INPUT:

- *i* – nonnegative integer
- *separating_element* – a separating element of the function field; the default is the generator of the rational function field

EXAMPLES:

```

sage: K.<t> = FunctionField(GF(2))
sage: f = t^2
sage: f.higher_derivative(2)
↪ # needs sage.rings.function_field
1

```

```

>>> from sage.all import *
>>> K = FunctionField(GF(Integer(2)), names=('t',)); (t,) = K._first_ngens(1)
>>> f = t**Integer(2)
>>> f.higher_derivative(Integer(2))
↪ # needs sage.rings.function_field
1

```

```

sage: K.<x> = FunctionField(GF(4)); _.<Y> = K[]
↪ # needs sage.rings.finite_rings
sage: L.<y> = K.extension(Y^2 + Y + x + 1/x)
↪ # needs sage.rings.finite_rings sage.rings.function_field

```

(continues on next page)

(continued from previous page)

```
sage: (y^3 + x).higher_derivative(2) #_
↳needs sage.rings.finite_rings sage.rings.function_field
1/x^3*y + (x^6 + x^4 + x^3 + x^2 + x + 1)/x^5
```

```
>>> from sage.all import *
>>> K = FunctionField(GF(Integer(4)), names=('x',)); (x,) = K._first_ngens(1);
↳ _ = K['Y']; (Y,) = _._first_ngens(1) # needs sage.rings.finite_rings
>>> L = K.extension(Y**Integer(2) + Y + x + Integer(1)/x, names=('y',)); (y,) =
↳ L._first_ngens(1) # needs sage.rings.finite_rings sage.rings.function_field
>>> (y**Integer(3) + x).higher_derivative(Integer(2))
↳ # needs sage.rings.finite_rings sage.rings.function_field
1/x^3*y + (x^6 + x^4 + x^3 + x^2 + x + 1)/x^5
```

is_integral()

Determine if the element is integral over the maximal order of the base field.

EXAMPLES:

```
sage: # needs sage.rings.function_field
sage: K.<x> = FunctionField(QQ); R.<y> = K[]
sage: L.<y> = K.extension(y^2 - x*y + 4*x^3)
sage: y.is_integral()
True
sage: (y/x).is_integral()
True
sage: (y/x)^2 - (y/x) + 4*x
0
sage: (y/x^2).is_integral()
False
sage: (y/x).minimal_polynomial('W')
W^2 - W + 4*x
```

```
>>> from sage.all import *
>>> # needs sage.rings.function_field
>>> K = FunctionField(QQ, names=('x',)); (x,) = K._first_ngens(1); R = K['y'];
↳ (y,) = R._first_ngens(1)
>>> L = K.extension(y**Integer(2) - x*y + Integer(4)*x**Integer(3), names=('y',)); (y,) = L._first_ngens(1)
>>> y.is_integral()
True
>>> (y/x).is_integral()
True
>>> (y/x)**Integer(2) - (y/x) + Integer(4)*x
0
>>> (y/x**Integer(2)).is_integral()
False
>>> (y/x).minimal_polynomial('W')
W^2 - W + 4*x
```

is_nth_power(n)

Return whether this element is an n-th power in the rational function field.

INPUT:

- n – integer

OUTPUT:

Returns `True` if there is an element a in the function field such that this element equals a^n .

See also

`nth_root()`

EXAMPLES:

```
sage: K.<x> = FunctionField(GF(3))
sage: f = (x+1)/(x-1)
sage: f.is_nth_power(2)
False
```

```
>>> from sage.all import *
>>> K = FunctionField(GF(Integer(3)), names=('x',)); (x,) = K._first_ngens(1)
>>> f = (x+Integer(1))/(x-Integer(1))
>>> f.is_nth_power(Integer(2))
False
```

matrix (*base=None*)

Return the matrix of multiplication by this element, interpreting this element as an element of a vector space over *base*.

INPUT:

- *base* – a function field (default: `None`); if `None`, then the matrix is formed over the base field of this function field

EXAMPLES:

A rational function field:

```
sage: K.<t> = FunctionField(QQ)
sage: t.matrix()                                     # _
↪needs sage.modules
[t]
sage: (1/(t+1)).matrix()                             # _
↪needs sage.modules
[1/(t + 1)]
```

```
>>> from sage.all import *
>>> K = FunctionField(QQ, names=('t',)); (t,) = K._first_ngens(1)
>>> t.matrix()                                     # _
↪needs sage.modules
[t]
>>> (Integer(1)/(t+Integer(1))).matrix()             _
↪                               # needs sage.modules
[1/(t + 1)]
```

Now an example in a nontrivial extension of a rational function field:

```

sage: # needs sage.rings.function_field
sage: K.<x> = FunctionField(QQ); R.<y> = K[]
sage: L.<y> = K.extension(y^2 - x*y + 4*x^3)
sage: y.matrix()
[      0      1]
[-4*x^3      x]
sage: y.matrix().charpoly('Z')
Z^2 - x*Z + 4*x^3

```

```

>>> from sage.all import *
>>> # needs sage.rings.function_field
>>> K = FunctionField(QQ, names=('x',)); (x,) = K._first_ngens(1); R = K['y'];
↪ (y,) = R._first_ngens(1)
>>> L = K.extension(y**Integer(2) - x*y + Integer(4)*x**Integer(3), names=('y
↪ ',)); (y,) = L._first_ngens(1)
>>> y.matrix()
[      0      1]
[-4*x^3      x]
>>> y.matrix().charpoly('Z')
Z^2 - x*Z + 4*x^3

```

An example in a relative extension, where neither function field is rational:

```

sage: # needs sage.rings.function_field
sage: K.<x> = FunctionField(QQ)
sage: R.<y> = K[]
sage: L.<y> = K.extension(y^2 - x*y + 4*x^3)
sage: M.<T> = L[]
sage: Z.<alpha> = L.extension(T^3 - y^2*T + x)
sage: alpha.matrix()
[      0      1      0]
[      0      0      1]
[-x x*y - 4*x^3      0]
sage: alpha.matrix(K)
[      0      0      1      0      0      ↪
↪ 0]
[      0      0      0      1      0      ↪
↪ 0]
[      0      0      0      0      1      ↪
↪ 0]
[      0      0      0      0      0      ↪
↪ 1]
[      -x      0      -4*x^3      x      0      ↪
↪ 0]
[      0      -x      -4*x^4 -4*x^3 + x^2      0      ↪
↪ 0]
sage: alpha.matrix(Z)
[alpha]

```

```

>>> from sage.all import *
>>> # needs sage.rings.function_field
>>> K = FunctionField(QQ, names=('x',)); (x,) = K._first_ngens(1)

```

(continues on next page)


```
>>> R = K['y']; (y,) = R._first_ngens(1)
>>> L = K.extension(y**Integer(2) - x*y + Integer(4)*x**Integer(3), names=('y'
↳ ',)); (y,) = L._first_ngens(1)
>>> M = L['T']; (T,) = M._first_ngens(1)
>>> Z = L.extension(T**Integer(3) - y**Integer(2)*T + x, names=('alpha',)); ↵
↳ (alpha,) = Z._first_ngens(1)
>>> alpha.matrix()
[
    0          1          0]
[
    0          0          1]
[
    -x x*y - 4*x^3      0]
>>> alpha.matrix(K)
[
    0          0          1          0          0          0          ↵
↳ 0]
[
    0          0          0          1          0          0          ↵
↳ 0]
[
    0          0          0          0          1          0          ↵
↳ 0]
[
    0          0          0          0          0          0          ↵
↳ 1]
[
    -x          0      -4*x^3          x          0          0          ↵
↳ 0]
[
    0          -x      -4*x^4 -4*x^3 + x^2          0          0          ↵
↳ 0]
>>> alpha.matrix(Z)
[alpha]
```

```
sage: # needs sage.rings.function_field
sage: K.<x> = FunctionField(QQ)
sage: R.<y> = K[]
sage: L.<y> = K.extension(y^5 - (x^3 + 2*x*y + 1/x))
sage: V, from_V, to_V = L.vector_space()
sage: y5 = to_V(y^5); y5
((x^4 + 1)/x, 2*x, 0, 0, 0)
sage: y4y = to_V(y^4) * y.matrix(); y4y
((x^4 + 1)/x, 2*x, 0, 0, 0)
sage: y5 == y4y
True
```

(continues on next page)

(continued from previous page)

True

minimal_polynomial(*args, **kws)

Return the minimal polynomial of the element. Give an optional input string to name the variable in the characteristic polynomial.

EXAMPLES:

```
sage: K.<x> = FunctionField(QQ); R.<y> = K[]
sage: x.minimal_polynomial('W')
↳needs sage.modules
W - x

sage: # needs sage.rings.function_field
sage: L.<y> = K.extension(y^2 - x*y + 4*x^3); R.<z> = L[]
sage: M.<z> = L.extension(z^3 - y^2*z + x)
sage: y.minimal_polynomial('W')
W^2 - x*W + 4*x^3
sage: z.minimal_polynomial('W')
W^3 + (-x*y + 4*x^3)*W + x
```

```
>>> from sage.all import *
>>> K = FunctionField(QQ, names=('x',)); (x,) = K._first_ngens(1); R = K['y'];
↳ (y,) = R._first_ngens(1)
>>> x.minimal_polynomial('W')
↳needs sage.modules
W - x

>>> # needs sage.rings.function_field
>>> L = K.extension(y**Integer(2) - x*y + Integer(4)*x**Integer(3), names=('y',)); (y,) = L._first_ngens(1); R = L['z']; (z,) = R._first_ngens(1)
>>> M = L.extension(z**Integer(3) - y**Integer(2)*z + x, names=('z',)); (z,) = M._first_ngens(1)
>>> y.minimal_polynomial('W')
W^2 - x*W + 4*x^3
>>> z.minimal_polynomial('W')
W^3 + (-x*y + 4*x^3)*W + x
```

minpoly(*args, **kws)

Return the minimal polynomial of the element. Give an optional input string to name the variable in the characteristic polynomial.

EXAMPLES:

```
sage: K.<x> = FunctionField(QQ); R.<y> = K[]
sage: x.minimal_polynomial('W')
↳needs sage.modules
W - x

sage: # needs sage.rings.function_field
sage: L.<y> = K.extension(y^2 - x*y + 4*x^3); R.<z> = L[]
sage: M.<z> = L.extension(z^3 - y^2*z + x)
sage: y.minimal_polynomial('W')
```

(continues on next page)

(continued from previous page)

```

W^2 - x*W + 4*x^3
sage: z.minimal_polynomial('W')
W^3 + (-x*y + 4*x^3)*W + x

```

```

>>> from sage.all import *
>>> K = FunctionField(QQ, names=('x',)); (x,) = K._first_ngens(1); R = K['y'];
↳ (y,) = R._first_ngens(1)
>>> x.minimal_polynomial('W') #↳
↳needs sage.modules
W - x

>>> # needs sage.rings.function_field
>>> L = K.extension(y**Integer(2) - x*y + Integer(4)*x**Integer(3), names=('y
↳ ',)); (y,) = L._first_ngens(1); R = L['z']; (z,) = R._first_ngens(1)
>>> M = L.extension(z**Integer(3) - y**Integer(2)*z + x, names=('z',)); (z,)↳
↳ M._first_ngens(1)
>>> y.minimal_polynomial('W')
W^2 - x*W + 4*x^3
>>> z.minimal_polynomial('W')
W^3 + (-x*y + 4*x^3)*W + x

```

norm()

Return the norm of the element.

EXAMPLES:

```

sage: K.<x> = FunctionField(QQ); R.<y> = K[]
sage: L.<y> = K.extension(y^2 - x*y + 4*x^3) #↳
↳needs sage.rings.function_field
sage: y.norm() #↳
↳needs sage.rings.function_field
4*x^3

```

```

>>> from sage.all import *
>>> K = FunctionField(QQ, names=('x',)); (x,) = K._first_ngens(1); R = K['y'];
↳ (y,) = R._first_ngens(1)
>>> L = K.extension(y**Integer(2) - x*y + Integer(4)*x**Integer(3), names=('y
↳ ',)); (y,) = L._first_ngens(1) # needs sage.rings.function_field
>>> y.norm() #↳
↳needs sage.rings.function_field
4*x^3

```

The norm is relative:

```

sage: K.<x> = FunctionField(QQ); R.<y> = K[]
sage: L.<y> = K.extension(y^2 - x*y + 4*x^3); R.<z> = L[] #↳
↳needs sage.rings.function_field
sage: M.<z> = L.extension(z^3 - y^2*z + x) #↳
↳needs sage.rings.function_field
sage: z.norm() #↳
↳needs sage.rings.function_field
-x

```

(continues on next page)

(continued from previous page)

```

sage: z.norm().parent()
↳needs sage.rings.function_field
Function field in y defined by y^2 - x*y + 4*x^3

>>> from sage.all import *
>>> K = FunctionField(QQ, names=('x',)); (x,) = K._first_ngens(1); R = K['y'];
↳ (y,) = R._first_ngens(1)
>>> L = K.extension(y**Integer(2) - x*y + Integer(4)*x**Integer(3), names=('y
↳ ',)); (y,) = L._first_ngens(1); R = L['z']; (z,) = R._first_ngens(1) # needs_
↳ sage.rings.function_field
>>> M = L.extension(z**Integer(3) - y**Integer(2)*z + x, names=('z',)); (z,)
↳ = M._first_ngens(1) # needs sage.rings.function_field
>>> z.norm()
↳needs sage.rings.function_field
-x
>>> z.norm().parent()
↳needs sage.rings.function_field
Function field in y defined by y^2 - x*y + 4*x^3

```

nth_root (*n*)

Return an *n*-th root of this element in the function field.

INPUT:

- *n* – integer

OUTPUT:

Returns an element *a* in the function field such that this element equals a^n . Raises an error if no such element exists.

See also

`is_nth_power()`

EXAMPLES:

```

sage: K.<x> = FunctionField(GF(3))
sage: R.<y> = K[]
sage: L.<y> = K.extension(y^2 - x)
↳needs sage.rings.function_field
sage: L(y^27).nth_root(27)
↳needs sage.rings.function_field
y

```

```

>>> from sage.all import *
>>> K = FunctionField(GF(Integer(3)), names=('x',)); (x,) = K._first_ngens(1)
>>> R = K['y']; (y,) = R._first_ngens(1)
>>> L = K.extension(y**Integer(2) - x, names=('y',)); (y,) = L._first_ngens(1)
↳ # needs sage.rings.function_field
>>> L(y**Integer(27)).nth_root(Integer(27))
↳ # needs sage.rings.function_field
y

```

poles()

Return the list of the poles of the element.

EXAMPLES:

```
sage: K.<x> = FunctionField(GF(2))
sage: f = 1/(x^3 + x^2 + x)
sage: f.poles()                                     #_
↳needs sage.libs.pari sage.modules
[Place (x), Place (x^2 + x + 1)]
```

```
>>> from sage.all import *
>>> K = FunctionField(GF(Integer(2)), names=('x',)); (x,) = K._first_ngens(1)
>>> f = Integer(1)/(x**Integer(3) + x**Integer(2) + x)
>>> f.poles()                                     #_
↳needs sage.libs.pari sage.modules
[Place (x), Place (x^2 + x + 1)]
```

```
sage: K.<x> = FunctionField(GF(4)); _.<Y> = K[]
↳needs sage.rings.finite_rings
sage: L.<y> = K.extension(Y^2 + Y + x + 1/x)
↳needs sage.rings.finite_rings sage.rings.function_field
sage: (x/y).poles()                               #_
↳needs sage.rings.finite_rings sage.rings.function_field
[Place (1/x, 1/x*y), Place (x + 1, x*y)]
```

```
>>> from sage.all import *
>>> K = FunctionField(GF(Integer(4)), names=('x',)); (x,) = K._first_ngens(1);
↳ _ = K['Y']; (Y,) = _.first_ngens(1) # needs sage.rings.finite_rings
>>> L = K.extension(Y**Integer(2) + Y + x + Integer(1)/x, names=('y',)); (y,)
↳ = L._first_ngens(1) # needs sage.rings.finite_rings sage.rings.function_field
>>> (x/y).poles()                                 #_
↳needs sage.rings.finite_rings sage.rings.function_field
[Place (1/x, 1/x*y), Place (x + 1, x*y)]
```

subs (*in_dict=None*, ***kwds*)

Substitute the given generators with given values while not touching other generators.

INPUT:

- *in_dict* – (optional) dictionary of inputs
- ***kwds* – named parameters

OUTPUT: new object if substitution is possible, otherwise *self*

EXAMPLES:

Basic substitution:

```
sage: K = GF(7)
sage: Kx.<x> = FunctionField(K)
sage: y = polygen(Kx)
sage: f = x^6 + 3; f
x^6 + 3
```

```
>>> from sage.all import *
>>> K = GF(Integer(7))
>>> Kx = FunctionField(K, names=('x',)); (x,) = Kx._first_ngens(1)
>>> y = polygen(Kx)
>>> f = x**Integer(6) + Integer(3); f
x^6 + 3
```

We also substitute the generators in any base fields:

```
sage: # needs sage.rings.function_field
sage: K.<x> = FunctionField(QQ)
sage: R.<y> = K[]
sage: L.<y> = K.extension(y^3 - (x^3 + 2*x*y + 1/x))
sage: S.<t> = L[]
sage: M.<t> = L.extension(t^2 - x*y)
sage: f = 7 * t + 3*x*y
sage: f.subs(t=9)
3*x*y + 63
sage: f.subs(x=2, y=4)
7*t + 24
sage: f.subs(t=1, x=2, y=3)
25
```

```
>>> from sage.all import *
>>> # needs sage.rings.function_field
>>> K = FunctionField(QQ, names=('x',)); (x,) = K._first_ngens(1)
>>> R = K['y']; (y,) = R._first_ngens(1)
>>> L = K.extension(y**Integer(3) - (x**Integer(3) + Integer(2)*x*y +
↳ Integer(1)/x), names=('y',)); (y,) = L._first_ngens(1)
>>> S = L['t']; (t,) = S._first_ngens(1)
>>> M = L.extension(t**Integer(2) - x*y, names=('t',)); (t,) = M._first_
↳ ngens(1)
>>> f = Integer(7) * t + Integer(3)*x*y
>>> f.subs(t=Integer(9))
3*x*y + 63
>>> f.subs(x=Integer(2), y=Integer(4))
7*t + 24
>>> f.subs(t=Integer(1), x=Integer(2), y=Integer(3))
25
```

Because of the possibility of extension fields, a generator to substitute must be specified:

```
sage: K.<x> = FunctionField(QQ)
sage: f = x
sage: f.subs(2)
Traceback (most recent call last):
...
TypeError: in_dict must be a dict

sage: # needs sage.rings.function_field
sage: R.<y> = K[]
sage: L.<y> = K.extension(y^3 - (x^3 + 2*x*y + 1/x))
sage: f = x + y
```

(continues on next page)

(continued from previous page)

```
sage: f.subs(0)
Traceback (most recent call last):
...
TypeError: in_dict must be a dict
```

```
>>> from sage.all import *
>>> K = FunctionField(QQ, names=('x',)); (x,) = K._first_ngens(1)
>>> f = x
>>> f.subs(Integer(2))
Traceback (most recent call last):
...
TypeError: in_dict must be a dict

>>> # needs sage.rings.function_field
>>> R = K['y']; (y,) = R._first_ngens(1)
>>> L = K.extension(y**Integer(3) - (x**Integer(3) + Integer(2)*x*y +
↳ Integer(1)/x), names=('y',)); (y,) = L._first_ngens(1)
>>> f = x + y
>>> f.subs(Integer(0))
Traceback (most recent call last):
...
TypeError: in_dict must be a dict
```

We can also substitute using dictionary syntax:

```
sage: # needs sage.rings.function_field
sage: K.<x> = FunctionField(QQ)
sage: R.<y> = K[]
sage: L.<y> = K.extension(y^3 - (x^3 + 2*x*y + 1/x))
sage: S.<t> = L[]
sage: M.<t> = L.extension(t^2 - x*y)
sage: f = x + y + t
sage: f.subs({x: 1, y: 3, t: 4})
8
sage: f.subs({x: 1, t: 4})
y + 5
```

```
>>> from sage.all import *
>>> # needs sage.rings.function_field
>>> K = FunctionField(QQ, names=('x',)); (x,) = K._first_ngens(1)
>>> R = K['y']; (y,) = R._first_ngens(1)
>>> L = K.extension(y**Integer(3) - (x**Integer(3) + Integer(2)*x*y +
↳ Integer(1)/x), names=('y',)); (y,) = L._first_ngens(1)
>>> S = L['t']; (t,) = S._first_ngens(1)
>>> M = L.extension(t**Integer(2) - x*y, names=('t',)); (t,) = M._first_
↳ ngens(1)
>>> f = x + y + t
>>> f.subs({x: Integer(1), y: Integer(3), t: Integer(4)})
8
>>> f.subs({x: Integer(1), t: Integer(4)})
y + 5
```

`trace()`

Return the trace of the element.

EXAMPLES:

```
sage: K.<x> = FunctionField(QQ); R.<y> = K[]
sage: L.<y> = K.extension(y^2 - x*y + 4*x^3) #_
↪needs sage.rings.function_field
sage: y.trace() #_
↪needs sage.rings.function_field
x
```

```
>>> from sage.all import *
>>> K = FunctionField(QQ, names=('x',)); (x,) = K._first_ngens(1); R = K['y'];
↪ (y,) = R._first_ngens(1)
>>> L = K.extension(y**Integer(2) - x*y + Integer(4)*x**Integer(3), names=('y',)); (y,) = L._first_ngens(1) # needs sage.rings.function_field
>>> y.trace() #_
↪needs sage.rings.function_field
x
```

valuation (*place*)

Return the valuation of the element at the place.

INPUT:

- *place* – a place of the function field

EXAMPLES:

```
sage: K.<x> = FunctionField(GF(2)); _.<Y> = K[]
sage: L.<y> = K.extension(Y^2 + Y + x + 1/x) #_
↪needs sage.rings.function_field
sage: p = L.places_infinite()[0] #_
↪needs sage.rings.function_field
sage: y.valuation(p) #_
↪needs sage.rings.function_field
-1
```

```
>>> from sage.all import *
>>> K = FunctionField(GF(Integer(2)), names=('x',)); (x,) = K._first_ngens(1);
↪ _ = K['Y']; (Y,) = _._first_ngens(1)
>>> L = K.extension(Y**Integer(2) + Y + x + Integer(1)/x, names=('y',)); (y,) = L._first_ngens(1) # needs sage.rings.function_field
>>> p = L.places_infinite()[Integer(0)] #_
↪ # needs sage.rings.function_field
>>> y.valuation(p) #_
↪needs sage.rings.function_field
-1
```

```
sage: # needs sage.rings.function_field
sage: K.<x> = FunctionField(QQ); _.<Y> = K[]
sage: L.<y> = K.extension(Y^2 + Y + x + 1/x)
sage: O = L.maximal_order()
sage: p = O.ideal(x - 1).place()
```

(continues on next page)

(continued from previous page)

```
sage: y.valuation(p)
0
```

```
>>> from sage.all import *
>>> # needs sage.rings.function_field
>>> K = FunctionField(QQ, names=('x',)); (x,) = K._first_ngens(1); _ = K['Y'];
↪ (Y,) = _._first_ngens(1)
>>> L = K.extension(Y**Integer(2) + Y + x + Integer(1)/x, names=('y',)); (y,) ↪
↪ = L._first_ngens(1)
>>> O = L.maximal_order()
>>> p = O.ideal(x - Integer(1)).place()
>>> y.valuation(p)
0
```

zeros()

Return the list of the zeros of the element.

EXAMPLES:

```
sage: K.<x> = FunctionField(GF(2))
sage: f = 1/(x^3 + x^2 + x)
sage: f.zeros()
↪ needs sage.libs.pari sage.modules #_
[Place (1/x)]
```

```
>>> from sage.all import *
>>> K = FunctionField(GF(Integer(2)), names=('x',)); (x,) = K._first_ngens(1)
>>> f = Integer(1)/(x**Integer(3) + x**Integer(2) + x)
>>> f.zeros()
↪ needs sage.libs.pari sage.modules #_
[Place (1/x)]
```

```
sage: K.<x> = FunctionField(GF(4)); _.<Y> = K[]
↪ needs sage.rings.finite_rings
sage: L.<y> = K.extension(Y^2 + Y + x + 1/x)
↪ needs sage.rings.finite_rings sage.rings.function_field #_
sage: (x/y).zeros()
↪ needs sage.rings.finite_rings sage.rings.function_field #_
[Place (x, x*y)]
```

```
>>> from sage.all import *
>>> K = FunctionField(GF(Integer(4)), names=('x',)); (x,) = K._first_ngens(1);
↪ _ = K['Y']; (Y,) = _._first_ngens(1) # needs sage.rings.finite_rings
>>> L = K.extension(Y**Integer(2) + Y + x + Integer(1)/x, names=('y',)); (y,) ↪
↪ = L._first_ngens(1) # needs sage.rings.finite_rings sage.rings.function_field
>>> (x/y).zeros()
↪ needs sage.rings.finite_rings sage.rings.function_field #_
[Place (x, x*y)]
```

`sage.rings.function_field.element.is_FunctionFieldElement(x)`Return True if x is any type of function field element.**EXAMPLES:**

```

sage: t = FunctionField(QQ, 't').gen()
sage: sage.rings.function_field.element.is_FunctionFieldElement(t)
doctest:warning...
DeprecationWarning: The function is_FunctionFieldElement is deprecated;
use '....parent() in FunctionFields()' instead.
See https://github.com/sagemath/sage/issues/38289 for details.
True
sage: sage.rings.function_field.element.is_FunctionFieldElement(0)
False

```

```

>>> from sage.all import *
>>> t = FunctionField(QQ, 't').gen()
>>> sage.rings.function_field.element.is_FunctionFieldElement(t)
doctest:warning...
DeprecationWarning: The function is_FunctionFieldElement is deprecated;
use '....parent() in FunctionFields()' instead.
See https://github.com/sagemath/sage/issues/38289 for details.
True
>>> sage.rings.function_field.element.is_FunctionFieldElement(Integer(0))
False

```

`sage.rings.function_field.element.make_FunctionFieldElement` (*parent*, *element_class*,
representing_element)

Used for unpickling FunctionFieldElement objects (and subclasses).

EXAMPLES:

```

sage: from sage.rings.function_field.element import make_FunctionFieldElement
sage: K.<x> = FunctionField(QQ)
sage: make_FunctionFieldElement(K, K.element_class, (x+1)/x)
(x + 1)/x

```

```

>>> from sage.all import *
>>> from sage.rings.function_field.element import make_FunctionFieldElement
>>> K = FunctionField(QQ, names=('x',)); (x,) = K._first_ngens(1)
>>> make_FunctionFieldElement(K, K.element_class, (x+Integer(1))/x)
(x + 1)/x

```

ELEMENTS OF FUNCTION FIELDS: RATIONAL

class sage.rings.function_field.element_rational.**FunctionFieldElement_rational**

Bases: *FunctionFieldElement*

Elements of a rational function field.

EXAMPLES:

```
sage: K.<t> = FunctionField(QQ); K
Rational function field in t over Rational Field
sage: t^2 + 3/2*t
t^2 + 3/2*t
sage: FunctionField(QQ, 't').gen()^3
t^3
```

```
>>> from sage.all import *
>>> K = FunctionField(QQ, names=('t',)); (t,) = K._first_ngens(1); K
Rational function field in t over Rational Field
>>> t**Integer(2) + Integer(3)/Integer(2)*t
t^2 + 3/2*t
>>> FunctionField(QQ, 't').gen()**Integer(3)
t^3
```

denominator()

Return the denominator of the rational function.

EXAMPLES:

```
sage: K.<t> = FunctionField(QQ)
sage: f = (t+1) / (t^2 - 1/3); f
(t + 1)/(t^2 - 1/3)
sage: f.denominator()
t^2 - 1/3
```

```
>>> from sage.all import *
>>> K = FunctionField(QQ, names=('t',)); (t,) = K._first_ngens(1)
>>> f = (t+Integer(1)) / (t**Integer(2) - Integer(1)/Integer(3)); f
(t + 1)/(t^2 - 1/3)
>>> f.denominator()
t^2 - 1/3
```

element()

Return the underlying fraction field element that represents the element.

EXAMPLES:

```

sage: K.<t> = FunctionField(GF(7))
sage: t.element()
t
sage: type(t.element())                                     #_
↳needs sage.libs.ntl
<... 'sage.rings.fraction_field_FpT.FpTElement'>

sage: # needs sage.rings.finite_rings
sage: K.<t> = FunctionField(GF(131101))
sage: t.element()
t
sage: type(t.element())
<... 'sage.rings.fraction_field_element.FractionFieldElement_1poly_field'>

```

```

>>> from sage.all import *
>>> K = FunctionField(GF(Integer(7)), names=('t',)); (t,) = K._first_ngens(1)
>>> t.element()
t
>>> type(t.element())                                     #_
↳needs sage.libs.ntl
<... 'sage.rings.fraction_field_FpT.FpTElement'>

>>> # needs sage.rings.finite_rings
>>> K = FunctionField(GF(Integer(131101)), names=('t',)); (t,) = K._first_
↳ngens(1)
>>> t.element()
t
>>> type(t.element())
<... 'sage.rings.fraction_field_element.FractionFieldElement_1poly_field'>

```

factor()

Factor the rational function.

EXAMPLES:

```

sage: # needs sage.libs.pari
sage: K.<t> = FunctionField(QQ)
sage: f = (t+1) / (t^2 - 1/3)
sage: f.factor()
(t + 1) * (t^2 - 1/3)^-1
sage: (7*f).factor()
(7) * (t + 1) * (t^2 - 1/3)^-1
sage: ((7*f).factor()).unit()
7
sage: (f^3).factor()
(t + 1)^3 * (t^2 - 1/3)^-3

```

```

>>> from sage.all import *
>>> # needs sage.libs.pari
>>> K = FunctionField(QQ, names=('t',)); (t,) = K._first_ngens(1)
>>> f = (t+Integer(1)) / (t**Integer(2) - Integer(1)/Integer(3))

```

(continues on next page)

(continued from previous page)

```

>>> f.factor()
(t + 1) * (t^2 - 1/3)^-1
>>> (Integer(7)*f).factor()
(7) * (t + 1) * (t^2 - 1/3)^-1
>>> ((Integer(7)*f).factor()).unit()
7
>>> (f**Integer(3)).factor()
(t + 1)^3 * (t^2 - 1/3)^-3

```

inverse_mod(I)

Return an inverse of the element modulo the integral ideal I , if I and the element together generate the unit ideal.

EXAMPLES:

```

sage: K.<x> = FunctionField(QQ)
sage: O = K.maximal_order(); I = O.ideal(x^2 + 1)
sage: t = O(x + 1).inverse_mod(I); t
-1/2*x + 1/2
sage: (t*(x+1) - 1) in I
True

```

```

>>> from sage.all import *
>>> K = FunctionField(QQ, names=('x',)); (x,) = K._first_ngens(1)
>>> O = K.maximal_order(); I = O.ideal(x**Integer(2) + Integer(1))
>>> t = O(x + Integer(1)).inverse_mod(I); t
-1/2*x + 1/2
>>> (t*(x+Integer(1)) - Integer(1)) in I
True

```

is_nth_power(n)

Return whether this element is an n -th power in the rational function field.

INPUT:

- n – integer

OUTPUT:

Returns `True` if there is an element a in the function field such that this element equals a^n .

ALGORITHM:

If n is a power of the characteristic of the field and the constant base field is perfect, then this uses the algorithm described in Lemma 3 of [GiTr1996].

See also

`nth_root()`

EXAMPLES:

```

sage: # needs sage.rings.finite_rings
sage: K.<x> = FunctionField(GF(3))

```

(continues on next page)

(continued from previous page)

```

sage: f = (x+1)/(x-1)
sage: f.is_nth_power(1)
True
sage: f.is_nth_power(3)                                     #_
↳needs sage.modules
False
sage: (f^3).is_nth_power(3)                                 #_
↳needs sage.modules
True
sage: (f^9).is_nth_power(-9)                               #_
↳needs sage.modules
True

```

```

>>> from sage.all import *
>>> # needs sage.rings.finite_rings
>>> K = FunctionField(GF(Integer(3)), names=('x',)); (x,) = K._first_ngens(1)
>>> f = (x+Integer(1))/(x-Integer(1))
>>> f.is_nth_power(Integer(1))
True
>>> f.is_nth_power(Integer(3))                               _
↳ # needs sage.modules
False
>>> (f**Integer(3)).is_nth_power(Integer(3))                 _
↳ # needs sage.modules
True
>>> (f**Integer(9)).is_nth_power(-Integer(9))               _
↳ # needs sage.modules
True

```

is_square()

Return whether the element is a square.

EXAMPLES:

```

sage: K.<t> = FunctionField(QQ)
sage: t.is_square()
False
sage: (t^2/4).is_square()
True
sage: f = 9 * (t+1)^6 / (t^2 - 2*t + 1); f.is_square()
True

sage: K.<t> = FunctionField(GF(5))
sage: (-t^2).is_square()                                     #_
↳needs sage.libs.pari
True
sage: (-t^2).sqrt()                                         #_
↳needs sage.libs.pari
2*t

```

```

>>> from sage.all import *
>>> K = FunctionField(QQ, names=('t',)); (t,) = K._first_ngens(1)

```

(continues on next page)

(continued from previous page)

```

>>> t.is_square()
False
>>> (t**Integer(2)/Integer(4)).is_square()
True
>>> f = Integer(9) * (t+Integer(1))**Integer(6) / (t**Integer(2) -
↳ Integer(2)*t + Integer(1)); f.is_square()
True

>>> K = FunctionField(GF(Integer(5)), names=('t',)); (t,) = K._first_ngens(1)
>>> (-t**Integer(2)).is_square()
↳ # needs sage.libs.pari
True
>>> (-t**Integer(2)).sqrt()
↳ # needs sage.libs.pari
2*t

```

list()

Return a list with just the element.

The list represents the element when the rational function field is viewed as a (one-dimensional) vector space over itself.

EXAMPLES:

```

sage: K.<t> = FunctionField(QQ)
sage: t.list()
[t]

```

```

>>> from sage.all import *
>>> K = FunctionField(QQ, names=('t',)); (t,) = K._first_ngens(1)
>>> t.list()
[t]

```

nth_root(n)

Return an n -th root of this element in the function field.

INPUT:

- n – integer

OUTPUT:

Returns an element a in the rational function field such that this element equals a^n . Raises an error if no such element exists.

ALGORITHM:

If n is a power of the characteristic of the field and the constant base field is perfect, then this uses the algorithm described in Corollary 3 of [GiTr1996].

See also

`is_nth_power()`

EXAMPLES:

```

sage: K.<x> = FunctionField(GF(3))
sage: f = (x+1)/(x+2)
sage: f.nth_root(1)
(x + 1)/(x + 2)
sage: f.nth_root(3)
Traceback (most recent call last):
...
ValueError: element is not an n-th power
sage: (f^3).nth_root(3) #_
↳needs sage.modules
(x + 1)/(x + 2)
sage: (f^9).nth_root(-9) #_
↳needs sage.modules
(x + 2)/(x + 1)

```

```

>>> from sage.all import *
>>> K = FunctionField(GF(Integer(3)), names=('x',)); (x,) = K._first_ngens(1)
>>> f = (x+Integer(1))/(x+Integer(2))
>>> f.nth_root(Integer(1))
(x + 1)/(x + 2)
>>> f.nth_root(Integer(3))
Traceback (most recent call last):
...
ValueError: element is not an n-th power
>>> (f**Integer(3)).nth_root(Integer(3)) #_
↳ # needs sage.modules
(x + 1)/(x + 2)
>>> (f**Integer(9)).nth_root(-Integer(9)) #_
↳ # needs sage.modules
(x + 2)/(x + 1)

```

numerator()

Return the numerator of the rational function.

EXAMPLES:

```

sage: K.<t> = FunctionField(QQ)
sage: f = (t+1) / (t^2 - 1/3); f
(t + 1)/(t^2 - 1/3)
sage: f.numerator()
t + 1

```

```

>>> from sage.all import *
>>> K = FunctionField(QQ, names=('t',)); (t,) = K._first_ngens(1)
>>> f = (t+Integer(1)) / (t**Integer(2) - Integer(1)/Integer(3)); f
(t + 1)/(t^2 - 1/3)
>>> f.numerator()
t + 1

```

sqrt(all=False)

Return the square root of the rational function.

EXAMPLES:


```
sage: K.<t> = FunctionField(QQ)
sage: f = t^2 - 2 + 1/t^2; f.sqrt()
(t^2 - 1)/t
sage: f = t^2; f.sqrt(all=True)
[t, -t]
```

```
>>> from sage.all import *
>>> K = FunctionField(QQ, names=('t',)); (t,) = K._first_ngens(1)
>>> f = t**Integer(2) - Integer(2) + Integer(1)/t**Integer(2); f.sqrt()
(t^2 - 1)/t
>>> f = t**Integer(2); f.sqrt(all=True)
[t, -t]
```

valuation(place)

Return the valuation of the rational function at the place.

Rational function field places are associated with irreducible polynomials.

INPUT:

- place – a place or an irreducible polynomial

EXAMPLES:

```
sage: K.<t> = FunctionField(QQ)
sage: f = (t - 1)^2*(t + 1)/(t^2 - 1/3)^3
sage: f.valuation(t - 1)
2
sage: f.valuation(t)
0
sage: f.valuation(t^2 - 1/3)
-3

sage: K.<x> = FunctionField(GF(2))
sage: p = K.places_finite()[0] #_
↪needs sage.libs.pari
sage: (1/x^2).valuation(p) #_
↪needs sage.libs.pari
-2
```

```
>>> from sage.all import *
>>> K = FunctionField(QQ, names=('t',)); (t,) = K._first_ngens(1)
>>> f = (t - Integer(1))**Integer(2)*(t + Integer(1))/(t**Integer(2) -
↪Integer(1)/Integer(3))**Integer(3)
>>> f.valuation(t - Integer(1))
2
>>> f.valuation(t)
0
>>> f.valuation(t**Integer(2) - Integer(1)/Integer(3))
-3

>>> K = FunctionField(GF(Integer(2)), names=('x',)); (x,) = K._first_ngens(1)
>>> p = K.places_finite()[Integer(0)] #_
↪ # needs sage.libs.pari
```

(continues on next page)

(continued from previous page)

```
>>> (Integer(1)/x**Integer(2)).valuation(p)
↩      # needs sage.libs.pari
-2
```

ELEMENTS OF FUNCTION FIELDS: EXTENSION

```
class sage.rings.function_field.element_polymod.FunctionFieldElement_polymod
```

Bases: *FunctionFieldElement*

Elements of a finite extension of a function field.

EXAMPLES:

```
sage: K.<x> = FunctionField(QQ); R.<y> = K[]
sage: L.<y> = K.extension(y^2 - x*y + 4*x^3)
sage: x*y + 1/x^3
x*y + 1/x^3
```

```
>>> from sage.all import *
>>> K = FunctionField(QQ, names=('x',)); (x,) = K._first_ngens(1); R = K['y']; (y,
↪) = R._first_ngens(1)
>>> L = K.extension(y**Integer(2) - x*y + Integer(4)*x**Integer(3), names=('y',));
↪ (y,) = L._first_ngens(1)
>>> x*y + Integer(1)/x**Integer(3)
x*y + 1/x^3
```

element()

Return the underlying polynomial that represents the element.

EXAMPLES:

```
sage: K.<x> = FunctionField(QQ); R.<T> = K[]
sage: L.<y> = K.extension(T^2 - x*T + 4*x^3)
sage: f = y/x^2 + x/(x^2+1); f
1/x^2*y + x/(x^2 + 1)
sage: f.element()
1/x^2*y + x/(x^2 + 1)
```

```
>>> from sage.all import *
>>> K = FunctionField(QQ, names=('x',)); (x,) = K._first_ngens(1); R = K['T'];
↪ (T,) = R._first_ngens(1)
>>> L = K.extension(T**Integer(2) - x*T + Integer(4)*x**Integer(3), names=('y
↪ ',)); (y,) = L._first_ngens(1)
>>> f = y/x**Integer(2) + x/(x**Integer(2)+Integer(1)); f
1/x^2*y + x/(x^2 + 1)
>>> f.element()
1/x^2*y + x/(x^2 + 1)
```

is_nth_power(*n*)Return whether this element is an *n*-th power in the function field.

INPUT:

- *n* – integer

ALGORITHM:

If *n* is a power of the characteristic of the field and the constant base field is perfect, then this uses the algorithm described in Proposition 12 of [GiTr1996].

See also`nth_root()`

EXAMPLES:

```
sage: # needs sage.rings.finite_rings
sage: K.<x> = FunctionField(GF(4))
sage: R.<y> = K[]
sage: L.<y> = K.extension(y^2 - x)
sage: y.is_nth_power(2)
False
sage: L(x).is_nth_power(2)
True
```

```
>>> from sage.all import *
>>> # needs sage.rings.finite_rings
>>> K = FunctionField(GF(Integer(4)), names=('x',)); (x,) = K._first_ngens(1)
>>> R = K['y']; (y,) = R._first_ngens(1)
>>> L = K.extension(y**Integer(2) - x, names=('y',)); (y,) = L._first_ngens(1)
>>> y.is_nth_power(Integer(2))
False
>>> L(x).is_nth_power(Integer(2))
True
```

list()

Return the list of the coefficients representing the element.

If the function field is $K[y]/(f(y))$, then return the coefficients of the reduced presentation of the element as a polynomial in $K[y]$.

EXAMPLES:

```
sage: K.<x> = FunctionField(QQ); R.<y> = K[]
sage: L.<y> = K.extension(y^2 - x*y + 4*x^3)
sage: a = ~(2*y + 1/x); a
(-1/8*x^2/(x^5 + 1/8*x^2 + 1/16))*y + (1/8*x^3 + 1/16*x)/(x^5 + 1/8*x^2 + 1/16)
↪16)
sage: a.list()
[(1/8*x^3 + 1/16*x)/(x^5 + 1/8*x^2 + 1/16), -1/8*x^2/(x^5 + 1/8*x^2 + 1/16)]
sage: (x*y).list()
[0, x]
```

```

>>> from sage.all import *
>>> K = FunctionField(QQ, names=('x',)); (x,) = K._first_ngens(1); R = K['y'];
↪ (y,) = R._first_ngens(1)
>>> L = K.extension(y**Integer(2) - x*y + Integer(4)*x**Integer(3), names=('y
↪ ',)); (y,) = L._first_ngens(1)
>>> a = ~(Integer(2)*y + Integer(1)/x); a
(-1/8*x^2/(x^5 + 1/8*x^2 + 1/16))*y + (1/8*x^3 + 1/16*x)/(x^5 + 1/8*x^2 + 1/
↪ 16)
>>> a.list()
[(1/8*x^3 + 1/16*x)/(x^5 + 1/8*x^2 + 1/16), -1/8*x^2/(x^5 + 1/8*x^2 + 1/16)]
>>> (x*y).list()
[0, x]

```

nth_root(*n*)

Return an n -th root of this element in the function field.

INPUT:

- n – integer

OUTPUT:

Returns an element a in the function field such that this element equals a^n . Raises an error if no such element exists.

ALGORITHM:

If n is a power of the characteristic of the field and the constant base field is perfect, then this uses the algorithm described in Proposition 12 of [GiTr1996].

See also

`is_nth_power()`

EXAMPLES:

```

sage: K.<x> = FunctionField(GF(3))
sage: R.<y> = K[]
sage: L.<y> = K.extension(y^2 - x)
sage: L(y^3).nth_root(3)
y
sage: L(y^9).nth_root(-9)
1/x*y

```

```

>>> from sage.all import *
>>> K = FunctionField(GF(Integer(3)), names=('x',)); (x,) = K._first_ngens(1)
>>> R = K['y']; (y,) = R._first_ngens(1)
>>> L = K.extension(y**Integer(2) - x, names=('y',)); (y,) = L._first_ngens(1)
>>> L(y**Integer(3)).nth_root(Integer(3))
y
>>> L(y**Integer(9)).nth_root(-Integer(9))
1/x*y

```

This also works for inseparable extensions:

```
sage: K.<x> = FunctionField(GF(3))
sage: R.<y> = K[]
sage: L.<y> = K.extension(y^3 - x^2)
sage: L(x).nth_root(3)^3
x
sage: L(x^9).nth_root(-27)^-27
x^9
```

```
>>> from sage.all import *
>>> K = FunctionField(GF(Integer(3)), names=('x',)); (x,) = K._first_ngens(1)
>>> R = K['y']; (y,) = R._first_ngens(1)
>>> L = K.extension(y**Integer(3) - x**Integer(2), names=('y',)); (y,) = L._
↳first_ngens(1)
>>> L(x).nth_root(Integer(3))**Integer(3)
x
>>> L(x**Integer(9)).nth_root(-Integer(27))**-Integer(27)
x^9
```

ORDERS OF FUNCTION FIELDS

An order of a function field is a subring that is, as a module over the base maximal order, finitely generated and of maximal rank n , where n is the extension degree of the function field. All orders are subrings of maximal orders.

A rational function field has two maximal orders: maximal finite order o and maximal infinite order o_∞ . The maximal order of a rational function field over constant field k is just the polynomial ring $o = k[x]$. The maximal infinite order is the set of rational functions whose denominator has degree greater than or equal to that of the numerator.

EXAMPLES:

```
sage: K.<x> = FunctionField(QQ)
sage: O = K.maximal_order()
sage: I = O.ideal(1/x); I
Ideal (1/x) of Maximal order of Rational function field in x over Rational Field
sage: 1/x in O
False
sage: Oinf = K.maximal_order_infinite()
sage: 1/x in Oinf
True
```

```
>>> from sage.all import *
>>> K = FunctionField(QQ, names=('x',)); (x,) = K._first_ngens(1)
>>> O = K.maximal_order()
>>> I = O.ideal(Integer(1)/x); I
Ideal (1/x) of Maximal order of Rational function field in x over Rational Field
>>> Integer(1)/x in O
False
>>> Oinf = K.maximal_order_infinite()
>>> Integer(1)/x in Oinf
True
```

In an extension of a rational function field, an order over the maximal finite order is called a finite order while an order over the maximal infinite order is called an infinite order. Thus a function field has one maximal finite order O and one maximal infinite order O_∞ . There are other non-maximal orders such as equation orders:

```
sage: # needs sage.rings.function_field
sage: K.<x> = FunctionField(GF(3)); R.<y> = K[]
sage: L.<y> = K.extension(y^3 - y - x)
sage: O = L.equation_order()
sage: 1/y in O
False
sage: x/y in O
True
```

```

>>> from sage.all import *
>>> # needs sage.rings.function_field
>>> K = FunctionField(GF(Integer(3)), names=('x',)); (x,) = K._first_ngens(1); R = K[
↳ 'y']; (y,) = R._first_ngens(1)
>>> L = K.extension(y**Integer(3) - y - x, names=('y',)); (y,) = L._first_ngens(1)
>>> O = L.equation_order()
>>> Integer(1)/y in O
False
>>> x/y in O
True

```

Sage provides an extensive functionality for computations in maximal orders of function fields. For example, you can decompose a prime ideal of a rational function field in an extension:

```

sage: K.<x> = FunctionField(GF(2)); _.<t> = K[]
sage: o = K.maximal_order()
sage: p = o.ideal(x + 1)
sage: p.is_prime()
↳needs sage.libs.pari
True

sage: # needs sage.rings.function_field
sage: F.<y> = K.extension(t^3 - x^2*(x^2 + x + 1)^2)
sage: O = F.maximal_order()
sage: O.decomposition(p)
[(Ideal (x + 1, y + 1) of Maximal order
 of Function field in y defined by y^3 + x^6 + x^4 + x^2, 1, 1),
 (Ideal (x + 1, (1/(x^3 + x^2 + x))*y^2 + y + 1) of Maximal order
 of Function field in y defined by y^3 + x^6 + x^4 + x^2, 2, 1)]

sage: # needs sage.rings.function_field
sage: p1, relative_degree, ramification_index = O.decomposition(p)[1]
sage: p1.parent()
Monoid of ideals of Maximal order of Function field in y
defined by y^3 + x^6 + x^4 + x^2
sage: relative_degree
2
sage: ramification_index
1

```

```

>>> from sage.all import *
>>> K = FunctionField(GF(Integer(2)), names=('x',)); (x,) = K._first_ngens(1); _ = K[
↳ 't']; (t,) = _._first_ngens(1)
>>> o = K.maximal_order()
>>> p = o.ideal(x + Integer(1))
>>> p.is_prime()
↳needs sage.libs.pari
True

>>> # needs sage.rings.function_field
>>> F = K.extension(t**Integer(3) - x**Integer(2)*(x**Integer(2) + x +
↳ Integer(1))*Integer(2), names=('y',)); (y,) = F._first_ngens(1)
>>> O = F.maximal_order()

```

(continues on next page)

(continued from previous page)

```

>>> O.decomposition(p)
[(Ideal (x + 1, y + 1) of Maximal order
 of Function field in y defined by y^3 + x^6 + x^4 + x^2, 1, 1),
 (Ideal (x + 1, (1/(x^3 + x^2 + x))*y^2 + y + 1) of Maximal order
 of Function field in y defined by y^3 + x^6 + x^4 + x^2, 2, 1)]

>>> # needs sage.rings.function_field
>>> p1, relative_degree, ramification_index = O.decomposition(p)[Integer(1)]
>>> p1.parent()
Monoid of ideals of Maximal order of Function field in y
defined by y^3 + x^6 + x^4 + x^2
>>> relative_degree
2
>>> ramification_index
1

```

When the base constant field is the algebraic field $\overline{\mathbb{Q}}$, the only prime ideals of the maximal order of the rational function field are linear polynomials.

```

sage: # needs sage.rings.function_field sage.rings.number_field
sage: K.<x> = FunctionField(QQbar)
sage: R.<y> = K[]
sage: L.<y> = K.extension(y^2 - (x^3-x^2))
sage: p = K.maximal_order().ideal(x)
sage: L.maximal_order().decomposition(p)
[(Ideal (1/x*y - 1) of Maximal order of Function field in y defined by y^2 - x^3 + x^
↪2,
 1,
 1),
 (Ideal (1/x*y + 1) of Maximal order of Function field in y defined by y^2 - x^3 + x^
↪2,
 1,
 1)]

```

```

>>> from sage.all import *
>>> # needs sage.rings.function_field sage.rings.number_field
>>> K = FunctionField(QQbar, names=('x',)); (x,) = K._first_ngens(1)
>>> R = K['y']; (y,) = R._first_ngens(1)
>>> L = K.extension(y**Integer(2) - (x**Integer(3)-x**Integer(2)), names=('y',)); (y,
↪) = L._first_ngens(1)
>>> p = K.maximal_order().ideal(x)
>>> L.maximal_order().decomposition(p)
[(Ideal (1/x*y - 1) of Maximal order of Function field in y defined by y^2 - x^3 + x^
↪2,
 1,
 1),
 (Ideal (1/x*y + 1) of Maximal order of Function field in y defined by y^2 - x^3 + x^
↪2,
 1,
 1)]

```

AUTHORS:

- William Stein (2010): initial version
- Maarten Derickx (2011-09-14): fixed `ideal_with_gens_over_base()` for rational function fields
- Julian Rueth (2011-09-14): added check in `_element_constructor_`
- Kwankyu Lee (2017-04-30): added maximal orders of global function fields
- Brent Baccala (2019-12-20): support orders in characteristic zero

```
class sage.rings.function_field.order.FunctionFieldMaximalOrder (field, ideal_class=<class
                                                                    'sage.rings.func-
                                                                    tion_field.ideal.FunctionField-
                                                                    Ideal'>, category=None)
```

Bases: `UniqueRepresentation`, `FunctionFieldOrder`

Base class of maximal orders of function fields.

```
class sage.rings.function_field.order.FunctionFieldMaximalOrderInfinite (field,
                                                                    ideal_class=<class
                                                                    'sage.rings.func-
                                                                    tion_field.ideal.Func-
                                                                    tionFieldIdeal'>,
                                                                    category=None)
```

Bases: `FunctionFieldMaximalOrder`, `FunctionFieldOrderInfinite`

Base class of maximal infinite orders of function fields.

```
class sage.rings.function_field.order.FunctionFieldOrder (field, ideal_class=<class 'sage.rings.func-
                                                                    tion_field.ideal.FunctionFieldIdeal'>,
                                                                    category=None)
```

Bases: `FunctionFieldOrder_base`

Base class for orders in function fields.

```
class sage.rings.function_field.order.FunctionFieldOrderInfinite (field, ideal_class=<class
                                                                    'sage.rings.func-
                                                                    tion_field.ideal.FunctionField-
                                                                    Ideal'>, category=None)
```

Bases: `FunctionFieldOrder_base`

Base class for infinite orders in function fields.

```
class sage.rings.function_field.order.FunctionFieldOrder_base (field, ideal_class=<class
                                                                    'sage.rings.func-
                                                                    tion_field.ideal.FunctionField-
                                                                    Ideal'>, category=None)
```

Bases: `CachedRepresentation`, `Parent`

Base class for orders in function fields.

INPUT:

- `field` – function field

EXAMPLES:

```
sage: F = FunctionField(QQ, 'y')
sage: F.maximal_order()
Maximal order of Rational function field in y over Rational Field
```

```
>>> from sage.all import *
>>> F = FunctionField(QQ, 'y')
>>> F.maximal_order()
Maximal order of Rational function field in y over Rational Field
```

fraction_field()

Return the function field to which the order belongs.

EXAMPLES:

```
sage: FunctionField(QQ, 'y').maximal_order().function_field()
Rational function field in y over Rational Field
```

```
>>> from sage.all import *
>>> FunctionField(QQ, 'y').maximal_order().function_field()
Rational function field in y over Rational Field
```

function_field()

Return the function field to which the order belongs.

EXAMPLES:

```
sage: FunctionField(QQ, 'y').maximal_order().function_field()
Rational function field in y over Rational Field
```

```
>>> from sage.all import *
>>> FunctionField(QQ, 'y').maximal_order().function_field()
Rational function field in y over Rational Field
```

ideal_monoid()

Return the monoid of ideals of the order.

EXAMPLES:

```
sage: FunctionField(QQ, 'y').maximal_order().ideal_monoid()
Monoid of ideals of Maximal order of Rational function field in y over_
↪Rational Field
```

```
>>> from sage.all import *
>>> FunctionField(QQ, 'y').maximal_order().ideal_monoid()
Monoid of ideals of Maximal order of Rational function field in y over_
↪Rational Field
```

is_field(proof=True)

Return False since orders are never fields.

EXAMPLES:

```
sage: FunctionField(QQ, 'y').maximal_order().is_field()
False
```

```
>>> from sage.all import *
>>> FunctionField(QQ, 'y').maximal_order().is_field()
False
```

is_noetherian()

Return True since orders in function fields are Noetherian.

EXAMPLES:

```
sage: FunctionField(QQ, 'y').maximal_order().is_noetherian()
True
```

```
>>> from sage.all import *
>>> FunctionField(QQ, 'y').maximal_order().is_noetherian()
True
```

is_subring(*other*)

Return True if the order is a subring of the other order.

INPUT:

- *other* – order of the function field or the field itself

EXAMPLES:

```
sage: F = FunctionField(QQ, 'y')
sage: O = F.maximal_order()
sage: O.is_subring(F)
True
```

```
>>> from sage.all import *
>>> F = FunctionField(QQ, 'y')
>>> O = F.maximal_order()
>>> O.is_subring(F)
True
```

ORDERS OF FUNCTION FIELDS: RATIONAL

```
class sage.rings.function_field.order_rational.FunctionFieldMaximalOrderInfinite_rational(field,  
                                                                                       cat-  
                                                                                       e-  
                                                                                       gory=None)
```

Bases: *FunctionFieldMaximalOrderInfinite*

Maximal infinite orders of rational function fields.

INPUT:

- `field` – a rational function field

EXAMPLES:

```
sage: K.<t> = FunctionField(GF(19)); K  
Rational function field in t over Finite Field of size 19  
sage: R = K.maximal_order_infinite(); R  
Maximal infinite order of Rational function field in t over Finite Field of size_  
↪19
```

```
>>> from sage.all import *  
>>> K = FunctionField(GF(Integer(19)), names=('t',)); (t,) = K._first_ngens(1); K  
Rational function field in t over Finite Field of size 19  
>>> R = K.maximal_order_infinite(); R  
Maximal infinite order of Rational function field in t over Finite Field of size_  
↪19
```

basis()

Return the basis of the maximal infinite order as a module over itself.

EXAMPLES:

```
sage: K.<t> = FunctionField(GF(19))  
sage: O = K.maximal_order_infinite()  
sage: O.basis()  
(1,)  
  
sage: K.<t> = FunctionField(QQ)  
sage: O = K.maximal_order_infinite()  
sage: O.basis()  
(1,)
```

```

>>> from sage.all import *
>>> K = FunctionField(GF(Integer(19)), names=('t',)); (t,) = K._first_ngens(1)
>>> O = K.maximal_order_infinite()
>>> O.basis()
(1,)

>>> K = FunctionField(QQ, names=('t',)); (t,) = K._first_ngens(1)
>>> O = K.maximal_order_infinite()
>>> O.basis()
(1,)

```

ideal(*gens)

Return the fractional ideal generated by gens.

INPUT:

- gens – elements of the function field

EXAMPLES:

```

sage: K.<x> = FunctionField(QQ)
sage: O = K.maximal_order_infinite()
sage: O.ideal(x)
Ideal (x) of Maximal infinite order of Rational function field in x
over Rational Field
sage: O.ideal([x, 1/x]) == O.ideal(x, 1/x) # multiple generators may be
↪given as a list
True
sage: O.ideal(x^3 + 1, x^3 + 6)
Ideal (x^3) of Maximal infinite order of Rational function field in x
over Rational Field
sage: I = O.ideal((x^2+1)*(x^3+1), (x^3+6)*(x^2+1)); I
Ideal (x^5) of Maximal infinite order of Rational function field in x
over Rational Field
sage: O.ideal(I)
Ideal (x^5) of Maximal infinite order of Rational function field in x
over Rational Field

```

```

>>> from sage.all import *
>>> K = FunctionField(QQ, names=('x',)); (x,) = K._first_ngens(1)
>>> O = K.maximal_order_infinite()
>>> O.ideal(x)
Ideal (x) of Maximal infinite order of Rational function field in x
over Rational Field
>>> O.ideal([x, Integer(1)/x]) == O.ideal(x, Integer(1)/x) # multiple
↪generators may be given as a list
True
>>> O.ideal(x**Integer(3) + Integer(1), x**Integer(3) + Integer(6))
Ideal (x^3) of Maximal infinite order of Rational function field in x
over Rational Field
>>> I = O.ideal((x**Integer(2)+Integer(1))*(x**Integer(3)+Integer(1)),
↪(x**Integer(3)+Integer(6))*(x**Integer(2)+Integer(1))); I
Ideal (x^5) of Maximal infinite order of Rational function field in x
over Rational Field

```

(continues on next page)

(continued from previous page)

```
>>> O.ideal(I)
Ideal (x^5) of Maximal infinite order of Rational function field in x
over Rational Field
```

prime_ideal()

Return the unique prime ideal of the order.

EXAMPLES:

```
sage: K.<t> = FunctionField(GF(19))
sage: O = K.maximal_order_infinite()
sage: O.prime_ideal()
Ideal (1/t) of Maximal infinite order of Rational function field in t
over Finite Field of size 19
```

```
>>> from sage.all import *
>>> K = FunctionField(GF(Integer(19)), names=('t',)); (t,) = K._first_ngens(1)
>>> O = K.maximal_order_infinite()
>>> O.prime_ideal()
Ideal (1/t) of Maximal infinite order of Rational function field in t
over Finite Field of size 19
```

class sage.rings.function_field.order_rational.**FunctionFieldMaximalOrder_rational**(*field*)Bases: *FunctionFieldMaximalOrder*

Maximal orders of rational function fields.

INPUT:

- *field* – a function field

EXAMPLES:

```
sage: K.<t> = FunctionField(GF(19)); K
Rational function field in t over Finite Field of size 19
sage: R = K.maximal_order(); R
Maximal order of Rational function field in t over Finite Field of size 19
```

```
>>> from sage.all import *
>>> K = FunctionField(GF(Integer(19)), names=('t',)); (t,) = K._first_ngens(1); K
Rational function field in t over Finite Field of size 19
>>> R = K.maximal_order(); R
Maximal order of Rational function field in t over Finite Field of size 19
```

basis()

Return the basis of the order as a module over the polynomial ring.

This is the tuple (1,).

EXAMPLES:

```
sage: K.<t> = FunctionField(GF(19))
sage: O = K.maximal_order()
sage: O.basis()
(1,)
```

```
>>> from sage.all import *
>>> K = FunctionField(GF(Integer(19)), names=('t',)); (t,) = K._first_ngens(1)
>>> O = K.maximal_order()
>>> O.basis()
(1,)
```

gen (*n=0*)

Return the *n*-th generator of the order.

Since there is only one generator *n* must be 0.

EXAMPLES:

```
sage: O = FunctionField(QQ, 'y').maximal_order()
sage: O.gen()
y
```

```
>>> from sage.all import *
>>> O = FunctionField(QQ, 'y').maximal_order()
>>> O.gen()
y
```

ideal (**gens*)

Return the fractional ideal generated by *gens*.

INPUT:

- *gens* – elements of the function field

EXAMPLES:

```
sage: K.<x> = FunctionField(QQ)
sage: O = K.maximal_order()
sage: O.ideal(x)
Ideal (x) of Maximal order of Rational function field in x over Rational Field
sage: O.ideal([x, 1/x]) == O.ideal(x, 1/x) # multiple generators may be
↳given as a list
True
sage: O.ideal(x^3 + 1, x^3 + 6)
Ideal (1) of Maximal order of Rational function field in x over Rational Field
sage: I = O.ideal((x^2+1)*(x^3+1), (x^3+6)*(x^2+1)); I
Ideal (x^2 + 1) of Maximal order of Rational function field in x over
↳Rational Field
sage: O.ideal(I)
Ideal (x^2 + 1) of Maximal order of Rational function field in x over
↳Rational Field
```

```
>>> from sage.all import *
>>> K = FunctionField(QQ, names=('x',)); (x,) = K._first_ngens(1)
>>> O = K.maximal_order()
>>> O.ideal(x)
Ideal (x) of Maximal order of Rational function field in x over Rational Field
>>> O.ideal([x, Integer(1)/x]) == O.ideal(x, Integer(1)/x) # multiple
↳generators may be given as a list
```

(continues on next page)

(continued from previous page)

```

True
>>> O.ideal(x**Integer(3) + Integer(1), x**Integer(3) + Integer(6))
Ideal (1) of Maximal order of Rational function field in x over Rational Field
>>> I = O.ideal((x**Integer(2)+Integer(1))*(x**Integer(3)+Integer(1)),
↪ (x**Integer(3)+Integer(6))*(x**Integer(2)+Integer(1))); I
Ideal (x^2 + 1) of Maximal order of Rational function field in x over
↪ Rational Field
>>> O.ideal(I)
Ideal (x^2 + 1) of Maximal order of Rational function field in x over
↪ Rational Field

```

ideal_with_gens_over_base(gens)

Return the fractional ideal with generators gens.

INPUT:

- gens – elements of the function field

EXAMPLES:

```

sage: # needs sage.rings.function_field
sage: K.<x> = FunctionField(QQ); R.<y> = K[]
sage: L.<y> = K.extension(y^2 - x^3 - 1)
sage: O = L.equation_order()
sage: O.ideal_with_gens_over_base([x^3 + 1, -y])
Ideal (x^3 + 1, -y) of Order in Function field in y defined by y^2 - x^3 - 1

```

```

>>> from sage.all import *
>>> # needs sage.rings.function_field
>>> K = FunctionField(QQ, names=('x',)); (x,) = K._first_ngens(1); R = K['y'];
↪ (y,) = R._first_ngens(1)
>>> L = K.extension(y**Integer(2) - x**Integer(3) - Integer(1), names=('y',));
↪ (y,) = L._first_ngens(1)
>>> O = L.equation_order()
>>> O.ideal_with_gens_over_base([x**Integer(3) + Integer(1), -y])
Ideal (x^3 + 1, -y) of Order in Function field in y defined by y^2 - x^3 - 1

```

ngens()

Return the number of generators of the order.

This is 1.

EXAMPLES:

```

sage: FunctionField(QQ, 'y').maximal_order().ngens()
1

```

```

>>> from sage.all import *
>>> FunctionField(QQ, 'y').maximal_order().ngens()
1

```


ORDERS OF FUNCTION FIELDS: BASIS

```
class sage.rings.function_field.order_basis.FunctionFieldOrderInfinite_basis (basis,
                                                                    check=True)
```

Bases: *FunctionFieldOrderInfinite*

Order given by a basis over the infinite maximal order of the base field.

INPUT:

- *basis* – elements of the function field
- *check* – boolean (default: True); if True, check the basis generates an order

EXAMPLES:

```
sage: K.<x> = FunctionField(GF(7)); R.<y> = K[]
sage: L.<y> = K.extension(y^4 + x*y + 4*x + 1) #_
↳needs sage.rings.function_field
sage: O = L.equation_order_infinite(); O #_
↳needs sage.rings.function_field
Infinite order in Function field in y defined by y^4 + x*y + 4*x + 1
```

```
>>> from sage.all import *
>>> K = FunctionField(GF(Integer(7)), names=('x',)); (x,) = K._first_ngens(1); R_
↳= K['y']; (y,) = R._first_ngens(1)
>>> L = K.extension(y**Integer(4) + x*y + Integer(4)*x + Integer(1), names=('y',
↳)); (y,) = L._first_ngens(1) # needs sage.rings.function_field
>>> O = L.equation_order_infinite(); O #_
↳needs sage.rings.function_field
Infinite order in Function field in y defined by y^4 + x*y + 4*x + 1
```

The basis only defines an order if the module it generates is closed under multiplication and contains the identity element (only checked when *check* is True):

```
sage: O = L.order_infinite_with_basis([1, y, 1/x^2*y^2, y^3]); O #_
↳needs sage.rings.function_field
Traceback (most recent call last):
...
ValueError: the module generated by basis (1, y, 1/x^2*y^2, y^3)
must be closed under multiplication
```

```
>>> from sage.all import *
>>> O = L.order_infinite_with_basis([Integer(1), y, Integer(1)/
```

(continues on next page)

(continued from previous page)

```

↪x**Integer(2)*y**Integer(2), y**Integer(3)]]; O                                # needs sage.
↪rings.function_field
Traceback (most recent call last):
...
ValueError: the module generated by basis (1, y, 1/x^2*y^2, y^3)
must be closed under multiplication

```

The basis also has to be linearly independent and of the same rank as the degree of the function field of its elements (only checked when check is True):

```

sage: O = L.order_infinite_with_basis([1, y, 1/x^2*y^2, 1 + y]); O                #_
↪needs sage.rings.function_field
Traceback (most recent call last):
...
ValueError: The given basis vectors must be linearly independent.

```

```

>>> from sage.all import *
>>> O = L.order_infinite_with_basis([Integer(1), y, Integer(1)/
↪x**Integer(2)*y**Integer(2), Integer(1) + y]); O                                # needs sage.
↪rings.function_field
Traceback (most recent call last):
...
ValueError: The given basis vectors must be linearly independent.

```

Note that 1 does not need to be an element of the basis, as long as it is in the module spanned by it:

```

sage: # needs sage.rings.function_field
sage: O = L.order_infinite_with_basis([1 + 1/x*y, 1/x*y, 1/x^2*y^2, 1/x^3*y^3]); O
Infinite order in Function field in y defined by y^4 + x*y + 4*x + 1
sage: O.basis()
(1/x*y + 1, 1/x*y, 1/x^2*y^2, 1/x^3*y^3)

```

```

>>> from sage.all import *
>>> # needs sage.rings.function_field
>>> O = L.order_infinite_with_basis([Integer(1) + Integer(1)/x*y, Integer(1)/x*y,
↪Integer(1)/x**Integer(2)*y**Integer(2), Integer(1)/
↪x**Integer(3)*y**Integer(3)]); O
Infinite order in Function field in y defined by y^4 + x*y + 4*x + 1
>>> O.basis()
(1/x*y + 1, 1/x*y, 1/x^2*y^2, 1/x^3*y^3)

```

basis()

Return a basis of this order over the maximal order of the base field.

EXAMPLES:

```

sage: K.<x> = FunctionField(GF(7)); R.<y> = K[]
sage: L.<y> = K.extension(y^4 + x*y + 4*x + 1)                                     #_
↪needs sage.rings.function_field
sage: O = L.equation_order()                                                       #_
↪needs sage.rings.function_field
sage: O.basis()                                                                    #_

```

(continues on next page)

(continued from previous page)

```
↪needs sage.rings.function_field
(1, y, y^2, y^3)
```

```
>>> from sage.all import *
>>> K = FunctionField(GF(Integer(7)), names=('x',)); (x,) = K._first_ngens(1);
↪ R = K['y']; (y,) = R._first_ngens(1)
>>> L = K.extension(y**Integer(4) + x*y + Integer(4)*x + Integer(1), names=('y'
↪ ',')); (y,) = L._first_ngens(1) # needs sage.rings.function_field
>>> O = L.equation_order() #_
↪needs sage.rings.function_field
>>> O.basis() #_
↪needs sage.rings.function_field
(1, y, y^2, y^3)
```

free_module()

Return the free module formed by the basis over the maximal order of the base field.

EXAMPLES:

```
sage: K.<x> = FunctionField(GF(7)); R.<y> = K[]
sage: L.<y> = K.extension(y^4 + x*y + 4*x + 1) #_
↪needs sage.rings.function_field
sage: O = L.equation_order() #_
↪needs sage.rings.function_field
sage: O.free_module() #_
↪needs sage.rings.function_field
Free module of degree 4 and rank 4 over Maximal order of Rational
function field in x over Finite Field of size 7
Echelon basis matrix:
[1 0 0 0]
[0 1 0 0]
[0 0 1 0]
[0 0 0 1]
```

```
>>> from sage.all import *
>>> K = FunctionField(GF(Integer(7)), names=('x',)); (x,) = K._first_ngens(1);
↪ R = K['y']; (y,) = R._first_ngens(1)
>>> L = K.extension(y**Integer(4) + x*y + Integer(4)*x + Integer(1), names=('y'
↪ ',')); (y,) = L._first_ngens(1) # needs sage.rings.function_field
>>> O = L.equation_order() #_
↪needs sage.rings.function_field
>>> O.free_module() #_
↪needs sage.rings.function_field
Free module of degree 4 and rank 4 over Maximal order of Rational
function field in x over Finite Field of size 7
Echelon basis matrix:
[1 0 0 0]
[0 1 0 0]
[0 0 1 0]
[0 0 0 1]
```

ideal(*gens)

Return the fractional ideal generated by the elements in gens.

INPUT:

- gens – list of generators or an ideal in a ring which coerces to this order

EXAMPLES:

```
sage: K.<y> = FunctionField(QQ)
sage: O = K.maximal_order()
sage: O.ideal(y)
Ideal (y) of Maximal order of Rational function field in y over Rational Field
sage: O.ideal([y, 1/y]) == O.ideal(y, 1/y) # multiple generators may be given
↪ as a list
True
```

```
>>> from sage.all import *
>>> K = FunctionField(QQ, names=('y',)); (y,) = K._first_ngens(1)
>>> O = K.maximal_order()
>>> O.ideal(y)
Ideal (y) of Maximal order of Rational function field in y over Rational Field
>>> O.ideal([y, Integer(1)/y]) == O.ideal(y, Integer(1)/y) # multiple
↪ generators may be given as a list
True
```

A fractional ideal of a nontrivial extension:

```
sage: K.<x> = FunctionField(QQ); R.<y> = K[]
sage: O = K.maximal_order_infinite()
sage: I = O.ideal(x^2 - 4)
sage: L.<y> = K.extension(y^2 - x^3 - 1) #
↪ needs sage.rings.function_field
sage: S = L.order_infinite_with_basis([1, 1/x^2*y]) #
↪ needs sage.rings.function_field
```

```
>>> from sage.all import *
>>> K = FunctionField(QQ, names=('x',)); (x,) = K._first_ngens(1); R = K['y'];
↪ (y,) = R._first_ngens(1)
>>> O = K.maximal_order_infinite()
>>> I = O.ideal(x**Integer(2) - Integer(4))
>>> L = K.extension(y**Integer(2) - x**Integer(3) - Integer(1), names=('y',));
↪ (y,) = L._first_ngens(1) # needs sage.rings.function_field
>>> S = L.order_infinite_with_basis([Integer(1), Integer(1)/x**Integer(2)*y])
↪ # needs sage.rings.function_field
```

ideal_with_gens_over_base(gens)

Return the fractional ideal with basis gens over the maximal order of the base field.

It is not checked that gens really generates an ideal.

INPUT:

- gens – list of elements that are a basis for the ideal over the maximal order of the base field

EXAMPLES:

We construct an ideal in a rational function field:

```

sage: K.<y> = FunctionField(QQ)
sage: O = K.maximal_order()
sage: I = O.ideal([y]); I
Ideal (y) of Maximal order of Rational function field in y over Rational Field
sage: I*I
Ideal (y^2) of Maximal order of Rational function field in y over Rational_
↪Field

```

```

>>> from sage.all import *
>>> K = FunctionField(QQ, names=('y',)); (y,) = K._first_ngens(1)
>>> O = K.maximal_order()
>>> I = O.ideal([y]); I
Ideal (y) of Maximal order of Rational function field in y over Rational Field
>>> I*I
Ideal (y^2) of Maximal order of Rational function field in y over Rational_
↪Field

```

We construct some ideals in a nontrivial function field:

```

sage: # needs sage.rings.function_field
sage: K.<x> = FunctionField(GF(7)); R.<y> = K[]
sage: L.<y> = K.extension(y^2 - x^3 - 1)
sage: O = L.equation_order(); O
Order in Function field in y defined by y^2 + 6*x^3 + 6
sage: I = O.ideal_with_gens_over_base([1, y]); I
Ideal (1) of Order in Function field in y defined by y^2 + 6*x^3 + 6
sage: I.module()
Free module of degree 2 and rank 2 over
Maximal order of Rational function field in x over Finite Field of size 7
Echelon basis matrix:
[1 0]
[0 1]

```

```

>>> from sage.all import *
>>> # needs sage.rings.function_field
>>> K = FunctionField(GF(Integer(7)), names=('x',)); (x,) = K._first_ngens(1);
↪ R = K['y']; (y,) = R._first_ngens(1)
>>> L = K.extension(y**Integer(2) - x**Integer(3) - Integer(1), names=('y',));
↪ (y,) = L._first_ngens(1)
>>> O = L.equation_order(); O
Order in Function field in y defined by y^2 + 6*x^3 + 6
>>> I = O.ideal_with_gens_over_base([Integer(1), y]); I
Ideal (1) of Order in Function field in y defined by y^2 + 6*x^3 + 6
>>> I.module()
Free module of degree 2 and rank 2 over
Maximal order of Rational function field in x over Finite Field of size 7
Echelon basis matrix:
[1 0]
[0 1]

```

There is no check if the resulting object is really an ideal:

```

sage: # needs sage.rings.function_field
sage: K.<x> = FunctionField(GF(7)); R.<y> = K[]
sage: L.<y> = K.extension(y^2 - x^3 - 1)
sage: O = L.equation_order()
sage: I = O.ideal_with_gens_over_base([y]); I
Ideal (y) of Order in Function field in y defined by y^2 + 6*x^3 + 6
sage: y in I
True
sage: y^2 in I
False

```

```

>>> from sage.all import *
>>> # needs sage.rings.function_field
>>> K = FunctionField(GF(Integer(7)), names=('x',)); (x,) = K._first_ngens(1);
↪ R = K['y']; (y,) = R._first_ngens(1)
>>> L = K.extension(y**Integer(2) - x**Integer(3) - Integer(1), names=('y',));
↪ (y,) = L._first_ngens(1)
>>> O = L.equation_order()
>>> I = O.ideal_with_gens_over_base([y]); I
Ideal (y) of Order in Function field in y defined by y^2 + 6*x^3 + 6
>>> y in I
True
>>> y**Integer(2) in I
False

```

polynomial()

Return the defining polynomial of the function field of which this is an order.

EXAMPLES:

```

sage: K.<x> = FunctionField(GF(7)); R.<y> = K[]
sage: L.<y> = K.extension(y^4 + x*y + 4*x + 1) #_
↪ needs sage.rings.function_field
sage: O = L.equation_order() #_
↪ needs sage.rings.function_field
sage: O.polynomial() #_
↪ needs sage.rings.function_field
y^4 + x*y + 4*x + 1

```

```

>>> from sage.all import *
>>> K = FunctionField(GF(Integer(7)), names=('x',)); (x,) = K._first_ngens(1);
↪ R = K['y']; (y,) = R._first_ngens(1)
>>> L = K.extension(y**Integer(4) + x*y + Integer(4)*x + Integer(1), names=('y',));
↪ (y,) = L._first_ngens(1) # needs sage.rings.function_field
>>> O = L.equation_order() #_
↪ needs sage.rings.function_field
>>> O.polynomial() #_
↪ needs sage.rings.function_field
y^4 + x*y + 4*x + 1

```

class sage.rings.function_field.order_basis.**FunctionFieldOrder_basis** (basis, check=True)

Bases: *FunctionFieldOrder*

Order given by a basis over the maximal order of the base field.

INPUT:

- `basis` – list of elements of the function field
- `check` – boolean (default: `True`); if `True`, check whether the module that `basis` generates forms an order

EXAMPLES:

```
sage: K.<x> = FunctionField(GF(7)); R.<y> = K[]
sage: L.<y> = K.extension(y^4 + x*y + 4*x + 1) #_
↪needs sage.rings.function_field
sage: O = L.equation_order(); O #_
↪needs sage.rings.function_field
Order in Function field in y defined by y^4 + x*y + 4*x + 1
```

```
>>> from sage.all import *
>>> K = FunctionField(GF(Integer(7)), names=('x',)); (x,) = K._first_ngens(1); R_
↪= K['y']; (y,) = R._first_ngens(1)
>>> L = K.extension(y**Integer(4) + x*y + Integer(4)*x + Integer(1), names=('y',
↪)); (y,) = L._first_ngens(1) # needs sage.rings.function_field
>>> O = L.equation_order(); O #_
↪needs sage.rings.function_field
Order in Function field in y defined by y^4 + x*y + 4*x + 1
```

The basis only defines an order if the module it generates is closed under multiplication and contains the identity element:

```
sage: K.<x> = FunctionField(QQ)
sage: R.<y> = K[]
sage: L.<y> = K.extension(y^5 - (x^3 + 2*x*y + 1/x)) #_
↪needs sage.rings.function_field
sage: y.is_integral() #_
↪needs sage.rings.function_field
False
sage: L.order(y) #_
↪needs sage.rings.function_field
Traceback (most recent call last):
...
ValueError: the module generated by basis (1, y, y^2, y^3, y^4)
must be closed under multiplication
```

```
>>> from sage.all import *
>>> K = FunctionField(QQ, names=('x',)); (x,) = K._first_ngens(1)
>>> R = K['y']; (y,) = R._first_ngens(1)
>>> L = K.extension(y**Integer(5) - (x**Integer(3) + Integer(2)*x*y + Integer(1)/
↪x), names=('y',)); (y,) = L._first_ngens(1) # needs sage.rings.function_field
>>> y.is_integral() #_
↪needs sage.rings.function_field
False
>>> L.order(y) #_
↪needs sage.rings.function_field
Traceback (most recent call last):
...
ValueError: the module generated by basis (1, y, y^2, y^3, y^4)
must be closed under multiplication
```

The basis also has to be linearly independent and of the same rank as the degree of the function field of its elements (only checked when `check` is `True`):

```
sage: # needs sage.rings.function_field
sage: L.order(L(x))
Traceback (most recent call last):
...
ValueError: basis (1, x, x^2, x^3, x^4) is not linearly independent
sage: from sage.rings.function_field.order_basis import FunctionFieldOrder_basis
sage: FunctionFieldOrder_basis((y,y,y^3,y^4,y^5))
Traceback (most recent call last):
...
ValueError: basis (y, y, y^3, y^4, 2*x*y + (x^4 + 1)/x) is not linearly_
↳ independent
```

```
>>> from sage.all import *
>>> # needs sage.rings.function_field
>>> L.order(L(x))
Traceback (most recent call last):
...
ValueError: basis (1, x, x^2, x^3, x^4) is not linearly independent
>>> from sage.rings.function_field.order_basis import FunctionFieldOrder_basis
>>> FunctionFieldOrder_basis((y,y,y**Integer(3),y**Integer(4),y**Integer(5)))
Traceback (most recent call last):
...
ValueError: basis (y, y, y^3, y^4, 2*x*y + (x^4 + 1)/x) is not linearly_
↳ independent
```

basis()

Return a basis of the order over the maximal order of the base field.

EXAMPLES:

```
sage: # needs sage.rings.function_field
sage: K.<x> = FunctionField(GF(7)); R.<y> = K[]
sage: L.<y> = K.extension(y^4 + x*y + 4*x + 1)
sage: O = L.equation_order()
sage: O.basis()
(1, y, y^2, y^3)
```

```
>>> from sage.all import *
>>> # needs sage.rings.function_field
>>> K = FunctionField(GF(Integer(7)), names=('x',)); (x,) = K._first_ngens(1);
↳ R = K['y']; (y,) = R._first_ngens(1)
>>> L = K.extension(y**Integer(4) + x*y + Integer(4)*x + Integer(1), names=('y
↳ ',)); (y,) = L._first_ngens(1)
>>> O = L.equation_order()
>>> O.basis()
(1, y, y^2, y^3)
```

coordinate_vector(e)

Return the coordinates of e with respect to the basis of the order.

INPUT:

- `e` – element of the order or the function field

EXAMPLES:

```
sage: # needs sage.rings.function_field
sage: K.<x> = FunctionField(GF(7)); R.<y> = K[]
sage: L.<y> = K.extension(y^4 + x*y + 4*x + 1)
sage: O = L.equation_order()
sage: f = (x + y)^3
sage: O.coordinate_vector(f)
(x^3, 3*x^2, 3*x, 1)
```

```
>>> from sage.all import *
>>> # needs sage.rings.function_field
>>> K = FunctionField(GF(Integer(7)), names=('x',)); (x,) = K._first_ngens(1);
↪ R = K['y']; (y,) = R._first_ngens(1)
>>> L = K.extension(y**Integer(4) + x*y + Integer(4)*x + Integer(1), names=('y
↪ ',)); (y,) = L._first_ngens(1)
>>> O = L.equation_order()
>>> f = (x + y)**Integer(3)
>>> O.coordinate_vector(f)
(x^3, 3*x^2, 3*x, 1)
```

free_module()

Return the free module formed by the basis over the maximal order of the base function field.

EXAMPLES:

```
sage: # needs sage.rings.function_field
sage: K.<x> = FunctionField(GF(7)); R.<y> = K[]
sage: L.<y> = K.extension(y^4 + x*y + 4*x + 1)
sage: O = L.equation_order()
sage: O.free_module()
Free module of degree 4 and rank 4 over Maximal order of Rational
function field in x over Finite Field of size 7
Echelon basis matrix:
[1 0 0 0]
[0 1 0 0]
[0 0 1 0]
[0 0 0 1]
```

```
>>> from sage.all import *
>>> # needs sage.rings.function_field
>>> K = FunctionField(GF(Integer(7)), names=('x',)); (x,) = K._first_ngens(1);
↪ R = K['y']; (y,) = R._first_ngens(1)
>>> L = K.extension(y**Integer(4) + x*y + Integer(4)*x + Integer(1), names=('y
↪ ',)); (y,) = L._first_ngens(1)
>>> O = L.equation_order()
>>> O.free_module()
Free module of degree 4 and rank 4 over Maximal order of Rational
function field in x over Finite Field of size 7
Echelon basis matrix:
[1 0 0 0]
[0 1 0 0]
```

(continues on next page)

(continued from previous page)

```
[0 0 1 0]
[0 0 0 1]
```

ideal(*gens)

Return the fractional ideal generated by the elements in gens.

INPUT:

- gens – list of generators or an ideal in a ring which coerces to this order

EXAMPLES:

```
sage: K.<y> = FunctionField(QQ)
sage: O = K.maximal_order()
sage: O.ideal(y)
Ideal (y) of Maximal order of Rational function field in y over Rational Field
sage: O.ideal([y, 1/y]) == O.ideal(y, 1/y) # multiple generators may be given
↪ as a list
True
```

```
>>> from sage.all import *
>>> K = FunctionField(QQ, names=('y',)); (y,) = K._first_ngens(1)
>>> O = K.maximal_order()
>>> O.ideal(y)
Ideal (y) of Maximal order of Rational function field in y over Rational Field
>>> O.ideal([y, Integer(1)/y]) == O.ideal(y, Integer(1)/y) # multiple
↪ generators may be given as a list
True
```

A fractional ideal of a nontrivial extension:

```
sage: K.<x> = FunctionField(GF(7)); R.<y> = K[]
sage: O = K.maximal_order()
sage: I = O.ideal(x^2 - 4)

sage: # needs sage.rings.function_field
sage: L.<y> = K.extension(y^2 - x^3 - 1)
sage: S = L.equation_order()
sage: S.ideal(1/y)
Ideal (1, (6/(x^3 + 1))*y) of
Order in Function field in y defined by y^2 + 6*x^3 + 6
sage: I2 = S.ideal(x^2 - 4); I2
Ideal (x^2 + 3) of Order in Function field in y defined by y^2 + 6*x^3 + 6
sage: I2 == S.ideal(I)
True
```

```
>>> from sage.all import *
>>> K = FunctionField(GF(Integer(7)), names=('x',)); (x,) = K._first_ngens(1);
↪ R = K['y']; (y,) = R._first_ngens(1)
>>> O = K.maximal_order()
>>> I = O.ideal(x**Integer(2) - Integer(4))

>>> # needs sage.rings.function_field
```

(continues on next page)

(continued from previous page)

```

>>> L = K.extension(y**Integer(2) - x**Integer(3) - Integer(1), names=('y',));
↪ (y,) = L._first_ngens(1)
>>> S = L.equation_order()
>>> S.ideal(Integer(1)/y)
Ideal (1, (6/(x^3 + 1))*y) of
Order in Function field in y defined by y^2 + 6*x^3 + 6
>>> I2 = S.ideal(x**Integer(2) - Integer(4)); I2
Ideal (x^2 + 3) of Order in Function field in y defined by y^2 + 6*x^3 + 6
>>> I2 == S.ideal(I)
True

```

ideal_with_gens_over_base (*gens*)

Return the fractional ideal with basis *gens* over the maximal order of the base field.

It is not checked that the *gens* really generates an ideal.

INPUT:

- *gens* – list of elements of the function field

EXAMPLES:

We construct an ideal in a rational function field:

```

sage: K.<y> = FunctionField(QQ)
sage: O = K.maximal_order()
sage: I = O.ideal([y]); I
Ideal (y) of Maximal order of Rational function field in y over Rational Field
sage: I * I
Ideal (y^2) of Maximal order of Rational function field in y over Rational_
↪Field

```

```

>>> from sage.all import *
>>> K = FunctionField(QQ, names=('y',)); (y,) = K._first_ngens(1)
>>> O = K.maximal_order()
>>> I = O.ideal([y]); I
Ideal (y) of Maximal order of Rational function field in y over Rational Field
>>> I * I
Ideal (y^2) of Maximal order of Rational function field in y over Rational_
↪Field

```

We construct some ideals in a nontrivial function field:

```

sage: # needs sage.rings.function_field
sage: K.<x> = FunctionField(GF(7)); R.<y> = K[]
sage: L.<y> = K.extension(y^2 - x^3 - 1)
sage: O = L.equation_order(); O
Order in Function field in y defined by y^2 + 6*x^3 + 6
sage: I = O.ideal_with_gens_over_base([1, y]); I
Ideal (1) of Order in Function field in y defined by y^2 + 6*x^3 + 6
sage: I.module()
Free module of degree 2 and rank 2 over
Maximal order of Rational function field in x over Finite Field of size 7
Echelon basis matrix:

```

(continues on next page)

(continued from previous page)

```
[1 0]
[0 1]
```

```
>>> from sage.all import *
>>> # needs sage.rings.function_field
>>> K = FunctionField(GF(Integer(7)), names=('x',)); (x,) = K._first_ngens(1);
↪ R = K['y']; (y,) = R._first_ngens(1)
>>> L = K.extension(y**Integer(2) - x**Integer(3) - Integer(1), names=('y',));
↪ (y,) = L._first_ngens(1)
>>> O = L.equation_order(); O
Order in Function field in y defined by y^2 + 6*x^3 + 6
>>> I = O.ideal_with_gens_over_base([Integer(1), y]); I
Ideal (1) of Order in Function field in y defined by y^2 + 6*x^3 + 6
>>> I.module()
Free module of degree 2 and rank 2 over
Maximal order of Rational function field in x over Finite Field of size 7
Echelon basis matrix:
[1 0]
[0 1]
```

There is no check if the resulting object is really an ideal:

```
sage: # needs sage.rings.function_field
sage: K.<x> = FunctionField(GF(7)); R.<y> = K[]
sage: L.<y> = K.extension(y^2 - x^3 - 1)
sage: O = L.equation_order()
sage: I = O.ideal_with_gens_over_base([y]); I
Ideal (y) of Order in Function field in y defined by y^2 + 6*x^3 + 6
sage: y in I
True
sage: y^2 in I
False
```

```
>>> from sage.all import *
>>> # needs sage.rings.function_field
>>> K = FunctionField(GF(Integer(7)), names=('x',)); (x,) = K._first_ngens(1);
↪ R = K['y']; (y,) = R._first_ngens(1)
>>> L = K.extension(y**Integer(2) - x**Integer(3) - Integer(1), names=('y',));
↪ (y,) = L._first_ngens(1)
>>> O = L.equation_order()
>>> I = O.ideal_with_gens_over_base([y]); I
Ideal (y) of Order in Function field in y defined by y^2 + 6*x^3 + 6
>>> y in I
True
>>> y**Integer(2) in I
False
```

polynomial()

Return the defining polynomial of the function field of which this is an order.

EXAMPLES:

```

sage: # needs sage.rings.function_field
sage: K.<x> = FunctionField(GF(7)); R.<y> = K[]
sage: L.<y> = K.extension(y^4 + x*y + 4*x + 1)
sage: O = L.equation_order()
sage: O.polynomial()
y^4 + x*y + 4*x + 1

```

```

>>> from sage.all import *
>>> # needs sage.rings.function_field
>>> K = FunctionField(GF(Integer(7)), names=('x',)); (x,) = K._first_ngens(1);
↪ R = K['y']; (y,) = R._first_ngens(1)
>>> L = K.extension(y**Integer(4) + x*y + Integer(4)*x + Integer(1), names=('y
↪ ',)); (y,) = L._first_ngens(1)
>>> O = L.equation_order()
>>> O.polynomial()
y^4 + x*y + 4*x + 1

```


ORDERS OF FUNCTION FIELDS: EXTENSION

```
class sage.rings.function_field.order_polymod.FunctionFieldMaximalOrderInfinite_polymod(field,
                                                                                       cat-
                                                                                       e-
                                                                                       gory=None)
```

Bases: *FunctionFieldMaximalOrderInfinite*

Maximal infinite orders of function fields.

INPUT:

- *field* – function field

EXAMPLES:

```
sage: K.<x> = FunctionField(GF(2)); _.<t> = PolynomialRing(K) #_
↳needs sage.rings.finite_rings
sage: F.<y> = K.extension(t^3 - x^2*(x^2+x+1)^2) #_
↳needs sage.rings.finite_rings
sage: F.maximal_order_infinite() #_
↳needs sage.rings.finite_rings
Maximal infinite order of Function field in y defined by y^3 + x^6 + x^4 + x^2

sage: K.<x> = FunctionField(GF(2)); _.<Y> = K[] #_
↳needs sage.rings.finite_rings
sage: L.<y> = K.extension(Y^2 + Y + x + 1/x) #_
↳needs sage.rings.finite_rings
sage: L.maximal_order_infinite() #_
↳needs sage.rings.finite_rings
Maximal infinite order of Function field in y defined by y^2 + y + (x^2 + 1)/x
```

```
>>> from sage.all import *
>>> K = FunctionField(GF(Integer(2)), names=('x',)); (x,) = K._first_ngens(1); _
↳= PolynomialRing(K, names=('t',)); (t,) = _._first_ngens(1) # needs sage.rings.
↳finite_rings
>>> F = K.extension(t**Integer(3) -_
↳x**Integer(2)*(x**Integer(2)+x+Integer(1))**Integer(2), names=('y',)); (y,) = F.
↳_first_ngens(1) # needs sage.rings.finite_rings
>>> F.maximal_order_infinite() #_
↳needs sage.rings.finite_rings
Maximal infinite order of Function field in y defined by y^3 + x^6 + x^4 + x^2

>>> K = FunctionField(GF(Integer(2)), names=('x',)); (x,) = K._first_ngens(1); _
```

(continues on next page)

(continued from previous page)

```

↪= K['Y']; (Y,) = _._first_ngens(1) # needs sage.rings.finite_rings
>>> L = K.extension(Y**Integer(2) + Y + x + Integer(1)/x, names=('y',)); (y,) = L.
↪_first_ngens(1) # needs sage.rings.finite_rings
>>> L.maximal_order_infinite() #_
↪needs sage.rings.finite_rings
Maximal infinite order of Function field in y defined by  $y^2 + y + (x^2 + 1)/x$ 

```

basis()

Return a basis of this order as a module over the maximal order of the base function field.

EXAMPLES:

```

sage: # needs sage.rings.finite_rings
sage: K.<x> = FunctionField(GF(2)); _.<t> = K[]
sage: L.<y> = K.extension(t^3 - x^2*(x^2 + x + 1)^2)
sage: Oinf = L.maximal_order_infinite()
sage: Oinf.basis()
(1, 1/x^2*y, (1/(x^4 + x^3 + x^2))*y^2)

```

```

>>> from sage.all import *
>>> # needs sage.rings.finite_rings
>>> K = FunctionField(GF(Integer(2)), names=('x',)); (x,) = K._first_ngens(1);
↪ _ = K['t']; (t,) = _._first_ngens(1)
>>> L = K.extension(t**Integer(3) - x**Integer(2)*(x**Integer(2) + x +
↪ Integer(1))**Integer(2), names=('y',)); (y,) = L._first_ngens(1)
>>> Oinf = L.maximal_order_infinite()
>>> Oinf.basis()
(1, 1/x^2*y, (1/(x^4 + x^3 + x^2))*y^2)

```

```

sage: # needs sage.rings.finite_rings
sage: K.<x> = FunctionField(GF(2)); _.<Y> = K[]
sage: L.<y> = K.extension(Y^2 + Y + x + 1/x)
sage: Oinf = L.maximal_order_infinite()
sage: Oinf.basis()
(1, 1/x*y)

```

```

>>> from sage.all import *
>>> # needs sage.rings.finite_rings
>>> K = FunctionField(GF(Integer(2)), names=('x',)); (x,) = K._first_ngens(1);
↪ _ = K['Y']; (Y,) = _._first_ngens(1)
>>> L = K.extension(Y**Integer(2) + Y + x + Integer(1)/x, names=('y',)); (y,) =
↪ L._first_ngens(1)
>>> Oinf = L.maximal_order_infinite()
>>> Oinf.basis()
(1, 1/x*y)

```

coordinate_vector(e)Return the coordinates of e with respect to the basis of the order.

INPUT:

- e – element of the function field

The returned coordinates are in the base maximal infinite order if and only if the element is in the order.

EXAMPLES:

```

sage: # needs sage.rings.finite_rings
sage: K.<x> = FunctionField(GF(2)); _.<Y> = K[]
sage: L.<y> = K.extension(Y^2 + Y + x + 1/x)
sage: Oinf = L.maximal_order_infinite()
sage: f = 1/y^2
sage: f in Oinf
True
sage: Oinf.coordinate_vector(f)
((x^3 + x^2 + x)/(x^4 + 1), x^3/(x^4 + 1))

```

```

>>> from sage.all import *
>>> # needs sage.rings.finite_rings
>>> K = FunctionField(GF(Integer(2)), names=('x',)); (x,) = K._first_ngens(1);
↪ _ = K['Y']; (Y,) = _._first_ngens(1)
>>> L = K.extension(Y**Integer(2) + Y + x + Integer(1)/x, names=('y',)); (y,) =
↪ L._first_ngens(1)
>>> Oinf = L.maximal_order_infinite()
>>> f = Integer(1)/y**Integer(2)
>>> f in Oinf
True
>>> Oinf.coordinate_vector(f)
((x^3 + x^2 + x)/(x^4 + 1), x^3/(x^4 + 1))

```

decomposition()

Return prime ideal decomposition of pO_∞ where p is the unique prime ideal of the maximal infinite order of the rational function field.

EXAMPLES:

```

sage: # needs sage.rings.finite_rings
sage: K.<x> = FunctionField(GF(2)); _.<t> = K[]
sage: F.<y> = K.extension(t^3 - x^2*(x^2 + x + 1)^2)
sage: Oinf = F.maximal_order_infinite()
sage: Oinf.decomposition()
[(Ideal ((1/(x^4 + x^3 + x^2))*y^2 + 1) of Maximal infinite order
of Function field in y defined by y^3 + x^6 + x^4 + x^2, 1, 1),
 (Ideal ((1/(x^4 + x^3 + x^2))*y^2 + 1/x^2*y + 1) of Maximal infinite order
of Function field in y defined by y^3 + x^6 + x^4 + x^2, 2, 1)]

```

```

>>> from sage.all import *
>>> # needs sage.rings.finite_rings
>>> K = FunctionField(GF(Integer(2)), names=('x',)); (x,) = K._first_ngens(1);
↪ _ = K['t']; (t,) = _._first_ngens(1)
>>> F = K.extension(t**Integer(3) - x**Integer(2)*(x**Integer(2) + x +
↪ Integer(1))**Integer(2), names=('y',)); (y,) = F._first_ngens(1)
>>> Oinf = F.maximal_order_infinite()
>>> Oinf.decomposition()
[(Ideal ((1/(x^4 + x^3 + x^2))*y^2 + 1) of Maximal infinite order
of Function field in y defined by y^3 + x^6 + x^4 + x^2, 1, 1),
 (Ideal ((1/(x^4 + x^3 + x^2))*y^2 + 1/x^2*y + 1) of Maximal infinite order
of Function field in y defined by y^3 + x^6 + x^4 + x^2, 2, 1)]

```

```

sage: # needs sage.rings.finite_rings
sage: K.<x> = FunctionField(GF(2)); _.<Y> = K[]
sage: L.<y> = K.extension(Y^2 + Y + x + 1/x)
sage: Oinf = L.maximal_order_infinite()
sage: Oinf.decomposition()
[(Ideal (1/x*y) of Maximal infinite order of Function field in y
defined by y^2 + y + (x^2 + 1)/x, 1, 2)]

```

```

>>> from sage.all import *
>>> # needs sage.rings.finite_rings
>>> K = FunctionField(GF(Integer(2)), names=('x',)); (x,) = K._first_ngens(1);
↪ _ = K['Y']; (Y,) = _._first_ngens(1)
>>> L = K.extension(Y**Integer(2) + Y + x + Integer(1)/x, names=('y',)); (y,) =
↪ L._first_ngens(1)
>>> Oinf = L.maximal_order_infinite()
>>> Oinf.decomposition()
[(Ideal (1/x*y) of Maximal infinite order of Function field in y
defined by y^2 + y + (x^2 + 1)/x, 1, 2)]

```

```

sage: K.<x> = FunctionField(QQ); _.<Y> = K[]
sage: F.<y> = K.extension(Y^3 - x^2*(x^2 + x + 1)^2)
sage: Oinf = F.maximal_order_infinite()
sage: Oinf.decomposition()
[(Ideal (1/x^2*y - 1) of Maximal infinite order
of Function field in y defined by y^3 - x^6 - 2*x^5 - 3*x^4 - 2*x^3 - x^2, 1,
↪ 1),
(Ideal ((1/(x^4 + x^3 + x^2))*y^2 + 1/x^2*y + 1) of Maximal infinite order
of Function field in y defined by y^3 - x^6 - 2*x^5 - 3*x^4 - 2*x^3 - x^2, 2,
↪ 1)]

```

```

>>> from sage.all import *
>>> K = FunctionField(QQ, names=('x',)); (x,) = K._first_ngens(1); _ = K['Y'];
↪ (Y,) = _._first_ngens(1)
>>> F = K.extension(Y**Integer(3) - x**Integer(2)*(x**Integer(2) + x +
↪ Integer(1))*Integer(2), names=('y',)); (y,) = F._first_ngens(1)
>>> Oinf = F.maximal_order_infinite()
>>> Oinf.decomposition()
[(Ideal (1/x^2*y - 1) of Maximal infinite order
of Function field in y defined by y^3 - x^6 - 2*x^5 - 3*x^4 - 2*x^3 - x^2, 1,
↪ 1),
(Ideal ((1/(x^4 + x^3 + x^2))*y^2 + 1/x^2*y + 1) of Maximal infinite order
of Function field in y defined by y^3 - x^6 - 2*x^5 - 3*x^4 - 2*x^3 - x^2, 2,
↪ 1)]

```

```

sage: K.<x> = FunctionField(QQ); _.<Y> = K[]
sage: L.<y> = K.extension(Y^2 + Y + x + 1/x)
sage: Oinf = L.maximal_order_infinite()
sage: Oinf.decomposition()
[(Ideal (1/x*y) of Maximal infinite order of Function field in y
defined by y^2 + y + (x^2 + 1)/x, 1, 2)]

```

```

>>> from sage.all import *
>>> K = FunctionField(QQ, names=('x',)); (x,) = K._first_ngens(1); _ = K['Y'];
↪ (Y,) = _._first_ngens(1)
>>> L = K.extension(Y**Integer(2) + Y + x + Integer(1)/x, names=('y',)); (y,) ↪
↪ = L._first_ngens(1)
>>> Oinf = L.maximal_order_infinite()
>>> Oinf.decomposition()
[(Ideal (1/x*y) of Maximal infinite order of Function field in y
defined by y^2 + y + (x^2 + 1)/x, 1, 2)]

```

different()

Return the different ideal of the maximal infinite order.

EXAMPLES:

```

sage: # needs sage.rings.finite_rings
sage: K.<x> = FunctionField(GF(2)); _.<Y> = K[]
sage: L.<y> = K.extension(Y^2 + Y + x + 1/x)
sage: Oinf = L.maximal_order_infinite()
sage: Oinf.different()
Ideal (1/x) of Maximal infinite order of Function field in y
defined by y^2 + y + (x^2 + 1)/x

```

```

>>> from sage.all import *
>>> # needs sage.rings.finite_rings
>>> K = FunctionField(GF(Integer(2)), names=('x',)); (x,) = K._first_ngens(1);
↪ _ = K['Y']; (Y,) = _._first_ngens(1)
>>> L = K.extension(Y**Integer(2) + Y + x + Integer(1)/x, names=('y',)); (y,) ↪
↪ = L._first_ngens(1)
>>> Oinf = L.maximal_order_infinite()
>>> Oinf.different()
Ideal (1/x) of Maximal infinite order of Function field in y
defined by y^2 + y + (x^2 + 1)/x

```

gen(n=0)

Return the n-th generator of the order.

The basis elements of the order are generators.

EXAMPLES:

```

sage: # needs sage.rings.finite_rings
sage: K.<x> = FunctionField(GF(2)); _.<t> = K[]
sage: L.<y> = K.extension(t^3 - x^2*(x^2 + x + 1)^2)
sage: Oinf = L.maximal_order_infinite()
sage: Oinf.gen()
1
sage: Oinf.gen(1)
1/x^2*y
sage: Oinf.gen(2)
(1/(x^4 + x^3 + x^2))*y^2
sage: Oinf.gen(3)
Traceback (most recent call last):

```

(continues on next page)

(continued from previous page)

```
...
IndexError: there are only 3 generators
```

```
>>> from sage.all import *
>>> # needs sage.rings.finite_rings
>>> K = FunctionField(GF(Integer(2)), names=('x',)); (x,) = K._first_ngens(1);
↪ _ = K['t']; (t,) = _._first_ngens(1)
>>> L = K.extension(t**Integer(3) - x**Integer(2)*(x**Integer(2) + x + ↪
↪ Integer(1))**Integer(2), names=('y',)); (y,) = L._first_ngens(1)
>>> Oinf = L.maximal_order_infinite()
>>> Oinf.gen()
1
>>> Oinf.gen(Integer(1))
1/x^2*y
>>> Oinf.gen(Integer(2))
(1/(x^4 + x^3 + x^2))*y^2
>>> Oinf.gen(Integer(3))
Traceback (most recent call last):
...
IndexError: there are only 3 generators
```

ideal(*gens)

Return the ideal generated by gens.

INPUT:

- gens – tuple of elements of the function field

EXAMPLES:

```
sage: # needs sage.rings.finite_rings
sage: K.<x> = FunctionField(GF(2)); _.<t> = K[]
sage: F.<y> = K.extension(t^3 - x^2*(x^2 + x + 1)^2)
sage: Oinf = F.maximal_order_infinite()
sage: I = Oinf.ideal(x, y); I
Ideal (y) of Maximal infinite order of Function field
in y defined by y^3 + x^6 + x^4 + x^2
```

```
>>> from sage.all import *
>>> # needs sage.rings.finite_rings
>>> K = FunctionField(GF(Integer(2)), names=('x',)); (x,) = K._first_ngens(1);
↪ _ = K['t']; (t,) = _._first_ngens(1)
>>> F = K.extension(t**Integer(3) - x**Integer(2)*(x**Integer(2) + x + ↪
↪ Integer(1))**Integer(2), names=('y',)); (y,) = F._first_ngens(1)
>>> Oinf = F.maximal_order_infinite()
>>> I = Oinf.ideal(x, y); I
Ideal (y) of Maximal infinite order of Function field
in y defined by y^3 + x^6 + x^4 + x^2
```

```
sage: # needs sage.rings.finite_rings
sage: K.<x> = FunctionField(GF(2)); _.<Y> = K[]
sage: L.<y> = K.extension(Y^2 + Y + x + 1/x)
sage: Oinf = L.maximal_order_infinite()
```

(continues on next page)

(continued from previous page)

```
sage: I = Oinf.ideal(x, y); I
Ideal (x) of Maximal infinite order of Function field
in y defined by y^2 + y + (x^2 + 1)/x
```

```
>>> from sage.all import *
>>> # needs sage.rings.finite_rings
>>> K = FunctionField(GF(Integer(2)), names=('x',)); (x,) = K._first_ngens(1);
↪ _ = K['Y']; (Y,) = _._first_ngens(1)
>>> L = K.extension(Y**Integer(2) + Y + x + Integer(1)/x, names=('y',)); (y,) =
↪ L._first_ngens(1)
>>> Oinf = L.maximal_order_infinite()
>>> I = Oinf.ideal(x, y); I
Ideal (x) of Maximal infinite order of Function field
in y defined by y^2 + y + (x^2 + 1)/x
```

ideal_with_gens_over_base(gens)

Return the ideal generated by gens as a module.

INPUT:

- gens – tuple of elements of the function field

EXAMPLES:

```
sage: K.<x> = FunctionField(GF(2)); R.<t> = K[]
sage: F.<y> = K.extension(t^3 - x^2*(x^2 + x + 1)^2)
sage: Oinf = F.maximal_order_infinite()
sage: Oinf.ideal_with_gens_over_base((x^2, y, (1/(x^2 + x + 1))*y^2))
Ideal (y) of Maximal infinite order of Function field in y
defined by y^3 + x^6 + x^4 + x^2
```

```
>>> from sage.all import *
>>> K = FunctionField(GF(Integer(2)), names=('x',)); (x,) = K._first_ngens(1);
↪ R = K['t']; (t,) = R._first_ngens(1)
>>> F = K.extension(t**Integer(3) - x**Integer(2)*(x**Integer(2) + x +
↪ Integer(1))**Integer(2), names=('y',)); (y,) = F._first_ngens(1)
>>> Oinf = F.maximal_order_infinite()
>>> Oinf.ideal_with_gens_over_base((x**Integer(2), y, (Integer(1)/
↪ (x**Integer(2) + x + Integer(1))*y**Integer(2)))
Ideal (y) of Maximal infinite order of Function field in y
defined by y^3 + x^6 + x^4 + x^2
```

ngens()

Return the number of generators of the order.

EXAMPLES:

```
sage: # needs sage.rings.finite_rings
sage: K.<x> = FunctionField(GF(2)); _.<t> = K[]
sage: L.<y> = K.extension(t^3 - x^2*(x^2 + x + 1)^2)
sage: Oinf = L.maximal_order_infinite()
sage: Oinf.ngens()
3
```

```

>>> from sage.all import *
>>> # needs sage.rings.finite_rings
>>> K = FunctionField(GF(Integer(2)), names=('x',)); (x,) = K._first_ngens(1);
↳ _ = K['t']; (t,) = _._first_ngens(1)
>>> L = K.extension(t**Integer(3) - x**Integer(2)*(x**Integer(2) + x +
↳ Integer(1))**Integer(2), names=('y',)); (y,) = L._first_ngens(1)
>>> Oinf = L.maximal_order_infinite()
>>> Oinf.ngens()
3

```

class sage.rings.function_field.order_polymod.**FunctionFieldMaximalOrder_global** (*field*)

Bases: *FunctionFieldMaximalOrder_polymod*

Maximal orders of global function fields.

INPUT:

- *field* – function field to which this maximal order belongs

EXAMPLES:

```

sage: K.<x> = FunctionField(GF(7)); R.<y> = K[] #_
↳ needs sage.rings.finite_rings
sage: L.<y> = K.extension(y^4 + x*y + 4*x + 1) #_
↳ needs sage.rings.finite_rings
sage: L.maximal_order() #_
↳ needs sage.rings.finite_rings
Maximal order of Function field in y defined by y^4 + x*y + 4*x + 1

```

```

>>> from sage.all import *
>>> K = FunctionField(GF(Integer(7)), names=('x',)); (x,) = K._first_ngens(1); R_
↳ = K['y']; (y,) = R._first_ngens(1) # needs sage.rings.finite_rings
>>> L = K.extension(y**Integer(4) + x*y + Integer(4)*x + Integer(1), names=('y',
↳ )); (y,) = L._first_ngens(1) # needs sage.rings.finite_rings
>>> L.maximal_order() #_
↳ needs sage.rings.finite_rings
Maximal order of Function field in y defined by y^4 + x*y + 4*x + 1

```

decomposition (*ideal*)

Return the decomposition of the prime ideal.

INPUT:

- *ideal* – prime ideal of the base maximal order

EXAMPLES:

```

sage: # needs sage.rings.finite_rings
sage: K.<x> = FunctionField(GF(2)); R.<t> = K[]
sage: F.<y> = K.extension(t^3 - x^2*(x^2 + x + 1)^2)
sage: o = K.maximal_order()
sage: O = F.maximal_order()
sage: p = o.ideal(x + 1)
sage: O.decomposition(p)
[(Ideal (x + 1, y + 1) of Maximal order
of Function field in y defined by y^3 + x^6 + x^4 + x^2, 1, 1),

```

(continues on next page)

(continued from previous page)

```
(Ideal (x + 1, (1/(x^3 + x^2 + x))*y^2 + y + 1) of Maximal order
of Function field in y defined by y^3 + x^6 + x^4 + x^2, 2, 1)]
```

```
>>> from sage.all import *
>>> # needs sage.rings.finite_rings
>>> K = FunctionField(GF(Integer(2)), names=('x',)); (x,) = K._first_ngens(1);
↪ R = K['t']; (t,) = R._first_ngens(1)
>>> F = K.extension(t**Integer(3) - x**Integer(2)*(x**Integer(2) + x +
↪ Integer(1))**Integer(2), names=('y',)); (y,) = F._first_ngens(1)
>>> o = K.maximal_order()
>>> O = F.maximal_order()
>>> p = o.ideal(x + Integer(1))
>>> O.decomposition(p)
[(Ideal (x + 1, y + 1) of Maximal order
of Function field in y defined by y^3 + x^6 + x^4 + x^2, 1, 1),
(Ideal (x + 1, (1/(x^3 + x^2 + x))*y^2 + y + 1) of Maximal order
of Function field in y defined by y^3 + x^6 + x^4 + x^2, 2, 1)]
```

p_radical (*prime*)

Return the prime-radical of the maximal order.

INPUT:

- *prime* – prime ideal of the maximal order of the base rational function field

The algorithm is outlined in Section 6.1.3 of [Coh1993].

EXAMPLES:

```
sage: # needs sage.rings.finite_rings
sage: K.<x> = FunctionField(GF(2)); _.<t> = K[]
sage: F.<y> = K.extension(t^3 - x^2 * (x^2 + x + 1)^2)
sage: o = K.maximal_order()
sage: O = F.maximal_order()
sage: p = o.ideal(x + 1)
sage: O.p_radical(p)
Ideal (x + 1) of Maximal order of Function field in y
defined by y^3 + x^6 + x^4 + x^2
```

```
>>> from sage.all import *
>>> # needs sage.rings.finite_rings
>>> K = FunctionField(GF(Integer(2)), names=('x',)); (x,) = K._first_ngens(1);
↪ _ = K['t']; (t,) = _._first_ngens(1)
>>> F = K.extension(t**Integer(3) - x**Integer(2) * (x**Integer(2) + x +
↪ Integer(1))**Integer(2), names=('y',)); (y,) = F._first_ngens(1)
>>> o = K.maximal_order()
>>> O = F.maximal_order()
>>> p = o.ideal(x + Integer(1))
>>> O.p_radical(p)
Ideal (x + 1) of Maximal order of Function field in y
defined by y^3 + x^6 + x^4 + x^2
```

```
class sage.rings.function_field.order_polymod.FunctionFieldMaximalOrder_polymod (field,
                                                                                   ideal_class=<class
                                                                                   'sage.rings.function_field.ideal_polymod.FunctionFieldIdeal_polymod'>)
```

Bases: *FunctionFieldMaximalOrder*

Maximal orders of extensions of function fields.

basis()

Return a basis of the order over the maximal order of the base function field.

EXAMPLES:

```
sage: # needs sage.rings.finite_rings
sage: K.<x> = FunctionField(GF(7)); R.<y> = K[]
sage: L.<y> = K.extension(y^4 + x*y + 4*x + 1)
sage: O = L.equation_order()
sage: O.basis()
(1, y, y^2, y^3)

sage: K.<x> = FunctionField(QQ)
sage: R.<t> = PolynomialRing(K)
sage: F.<y> = K.extension(t^4 + x^12*t^2 + x^18*t + x^21 + x^18)
sage: O = F.maximal_order()
sage: O.basis()
(1, 1/x^4*y, 1/x^9*y^2, 1/x^13*y^3)
```

```
>>> from sage.all import *
>>> # needs sage.rings.finite_rings
>>> K = FunctionField(GF(Integer(7)), names=('x',)); (x,) = K._first_ngens(1);
↳ R = K['y']; (y,) = R._first_ngens(1)
>>> L = K.extension(y**Integer(4) + x*y + Integer(4)*x + Integer(1), names=('y',)); (y,) = L._first_ngens(1)
↳
>>> O = L.equation_order()
>>> O.basis()
(1, y, y^2, y^3)

>>> K = FunctionField(QQ, names=('x',)); (x,) = K._first_ngens(1)
>>> R = PolynomialRing(K, names=('t',)); (t,) = R._first_ngens(1)
>>> F = K.extension(t**Integer(4) + x**Integer(12)*t**Integer(2) +
↳ x**Integer(18)*t + x**Integer(21) + x**Integer(18), names=('y',)); (y,) = F._first_ngens(1)
↳
>>> O = F.maximal_order()
>>> O.basis()
(1, 1/x^4*y, 1/x^9*y^2, 1/x^13*y^3)
```

codifferent()

Return the codifferent ideal of the function field.

EXAMPLES:

```

sage: # needs sage.rings.finite_rings
sage: K.<x> = FunctionField(GF(7)); R.<y> = K[]
sage: L.<y> = K.extension(y^4 + x*y + 4*x + 1)
sage: O = L.maximal_order()
sage: O.codifferent()
Ideal (1, (1/(x^4 + 4*x^3 + 3*x^2 + 6*x + 4))*y^3
+ ((5*x^3 + 6*x^2 + x + 6)/(x^4 + 4*x^3 + 3*x^2 + 6*x + 4))*y^2
+ ((x^3 + 2*x^2 + 2*x + 2)/(x^4 + 4*x^3 + 3*x^2 + 6*x + 4))*y
+ 6*x/(x^4 + 4*x^3 + 3*x^2 + 6*x + 4)) of Maximal order of Function field
in y defined by y^4 + x*y + 4*x + 1

```

```

>>> from sage.all import *
>>> # needs sage.rings.finite_rings
>>> K = FunctionField(GF(Integer(7)), names=('x',)); (x,) = K._first_ngens(1);
↪ R = K['y']; (y,) = R._first_ngens(1)
>>> L = K.extension(y**Integer(4) + x*y + Integer(4)*x + Integer(1), names=('y'
↪ ',)); (y,) = L._first_ngens(1)
>>> O = L.maximal_order()
>>> O.codifferent()
Ideal (1, (1/(x^4 + 4*x^3 + 3*x^2 + 6*x + 4))*y^3
+ ((5*x^3 + 6*x^2 + x + 6)/(x^4 + 4*x^3 + 3*x^2 + 6*x + 4))*y^2
+ ((x^3 + 2*x^2 + 2*x + 2)/(x^4 + 4*x^3 + 3*x^2 + 6*x + 4))*y
+ 6*x/(x^4 + 4*x^3 + 3*x^2 + 6*x + 4)) of Maximal order of Function field
in y defined by y^4 + x*y + 4*x + 1

```

coordinate_vector(e)

Return the coordinates of e with respect to the basis of this order.

EXAMPLES:

```

sage: # needs sage.rings.finite_rings
sage: K.<x> = FunctionField(GF(7)); R.<y> = K[]
sage: L.<y> = K.extension(y^4 + x*y + 4*x + 1)
sage: O = L.maximal_order()
sage: O.coordinate_vector(y)
(0, 1, 0, 0)
sage: O.coordinate_vector(x*y)
(0, x, 0, 0)

sage: K.<x> = FunctionField(QQ); R.<y> = K[]
sage: L.<y> = K.extension(y^4 + x*y + 4*x + 1)
sage: O = L.equation_order()
sage: f = (x + y)^3
sage: O.coordinate_vector(f)
(x^3, 3*x^2, 3*x, 1)

```

```

>>> from sage.all import *
>>> # needs sage.rings.finite_rings
>>> K = FunctionField(GF(Integer(7)), names=('x',)); (x,) = K._first_ngens(1);
↪ R = K['y']; (y,) = R._first_ngens(1)
>>> L = K.extension(y**Integer(4) + x*y + Integer(4)*x + Integer(1), names=('y'
↪ ',)); (y,) = L._first_ngens(1)
>>> O = L.maximal_order()

```

(continues on next page)

(continued from previous page)

```

>>> O.coordinate_vector(y)
(0, 1, 0, 0)
>>> O.coordinate_vector(x*y)
(0, x, 0, 0)

>>> K = FunctionField(QQ, names=('x',)); (x,) = K._first_ngens(1); R = K['y'];
↪ (y,) = R._first_ngens(1)
>>> L = K.extension(y**Integer(4) + x*y + Integer(4)*x + Integer(1), names=('y'
↪ ',)); (y,) = L._first_ngens(1)
>>> O = L.equation_order()
>>> f = (x + y)**Integer(3)
>>> O.coordinate_vector(f)
(x^3, 3*x^2, 3*x, 1)

```

decomposition(ideal)

Return the decomposition of the prime ideal.

INPUT:

- `ideal` – prime ideal of the base maximal order

EXAMPLES:

```

sage: # needs sage.rings.finite_rings
sage: K.<x> = FunctionField(GF(2)); R.<t> = K[]
sage: F.<y> = K.extension(t^3 - x^2*(x^2 + x + 1)^2)
sage: o = K.maximal_order()
sage: O = F.maximal_order()
sage: p = o.ideal(x + 1)
sage: O.decomposition(p)
[(Ideal (x + 1, y + 1) of Maximal order
of Function field in y defined by y^3 + x^6 + x^4 + x^2, 1, 1),
 (Ideal (x + 1, (1/(x^3 + x^2 + x))*y^2 + y + 1) of Maximal order
of Function field in y defined by y^3 + x^6 + x^4 + x^2, 2, 1)]

```

```

>>> from sage.all import *
>>> # needs sage.rings.finite_rings
>>> K = FunctionField(GF(Integer(2)), names=('x',)); (x,) = K._first_ngens(1);
↪ R = K['t']; (t,) = R._first_ngens(1)
>>> F = K.extension(t**Integer(3) - x**Integer(2)*(x**Integer(2) + x +
↪ Integer(1))**Integer(2), names=('y',)); (y,) = F._first_ngens(1)
>>> o = K.maximal_order()
>>> O = F.maximal_order()
>>> p = o.ideal(x + Integer(1))
>>> O.decomposition(p)
[(Ideal (x + 1, y + 1) of Maximal order
of Function field in y defined by y^3 + x^6 + x^4 + x^2, 1, 1),
 (Ideal (x + 1, (1/(x^3 + x^2 + x))*y^2 + y + 1) of Maximal order
of Function field in y defined by y^3 + x^6 + x^4 + x^2, 2, 1)]

```

ALGORITHM:

In principle, we're trying to compute a primary decomposition of the extension of `ideal` in `self` (an order, and therefore a ring). However, while we have primary decomposition methods for polynomial rings, we lack

any such method for an order. Therefore, we construct `self mod ideal` as a finite-dimensional algebra, a construct for which we do support primary decomposition.

See [Issue #28094](#) and <https://github.com/sagemath/sage/files/10659303/decomposition.pdf.gz>

Todo

Use Kummer's theorem to shortcut this code if possible, like as done in `FunctionFieldMaximalOrder_global.decomposition()`

`different()`

Return the different ideal of the function field.

EXAMPLES:

```
sage: # needs sage.rings.finite_rings
sage: K.<x> = FunctionField(GF(7)); R.<y> = K[]
sage: L.<y> = K.extension(y^4 + x*y + 4*x + 1)
sage: O = L.maximal_order()
sage: O.different()
Ideal (y^3 + 2*x)
of Maximal order of Function field in y defined by y^4 + x*y + 4*x + 1
```

```
>>> from sage.all import *
>>> # needs sage.rings.finite_rings
>>> K = FunctionField(GF(Integer(7)), names=('x',)); (x,) = K._first_ngens(1);
↳ R = K['y']; (y,) = R._first_ngens(1)
>>> L = K.extension(y**Integer(4) + x*y + Integer(4)*x + Integer(1), names=('y'
↳ ',)); (y,) = L._first_ngens(1)
>>> O = L.maximal_order()
>>> O.different()
Ideal (y^3 + 2*x)
of Maximal order of Function field in y defined by y^4 + x*y + 4*x + 1
```

`free_module()`

Return the free module formed by the basis over the maximal order of the base field.

EXAMPLES:

```
sage: # needs sage.rings.finite_rings
sage: K.<x> = FunctionField(GF(7)); R.<y> = K[]
sage: L.<y> = K.extension(y^4 + x*y + 4*x + 1)
sage: O = L.maximal_order()
sage: O.free_module()
Free module of degree 4 and rank 4 over
Maximal order of Rational function field in x over Finite Field of size 7
User basis matrix:
[1 0 0 0]
[0 1 0 0]
[0 0 1 0]
[0 0 0 1]
```

```

>>> from sage.all import *
>>> # needs sage.rings.finite_rings
>>> K = FunctionField(GF(Integer(7)), names=('x',)); (x,) = K._first_ngens(1);
↪ R = K['y']; (y,) = R._first_ngens(1)
>>> L = K.extension(y**Integer(4) + x*y + Integer(4)*x + Integer(1), names=('y'
↪ ',)); (y,) = L._first_ngens(1)
>>> O = L.maximal_order()
>>> O.free_module()
Free module of degree 4 and rank 4 over
Maximal order of Rational function field in x over Finite Field of size 7
User basis matrix:
[1 0 0 0]
[0 1 0 0]
[0 0 1 0]
[0 0 0 1]

```

gen ($n=0$)

Return the n -th generator of the order.

The basis elements of the order are generators.

EXAMPLES:

```

sage: # needs sage.rings.finite_rings
sage: K.<x> = FunctionField(GF(2)); _.<t> = K[]
sage: L.<y> = K.extension(t^3 - x^2*(x^2 + x + 1)^2)
sage: O = L.maximal_order()
sage: O.gen()
1
sage: O.gen(1)
y
sage: O.gen(2)
(1/(x^3 + x^2 + x))*y^2
sage: O.gen(3)
Traceback (most recent call last):
...
IndexError: there are only 3 generators

```

```

>>> from sage.all import *
>>> # needs sage.rings.finite_rings
>>> K = FunctionField(GF(Integer(2)), names=('x',)); (x,) = K._first_ngens(1);
↪ _ = K['t']; (t,) = _._first_ngens(1)
>>> L = K.extension(t**Integer(3) - x**Integer(2)*(x**Integer(2) + x +
↪ Integer(1))**Integer(2), names=('y',)); (y,) = L._first_ngens(1)
>>> O = L.maximal_order()
>>> O.gen()
1
>>> O.gen(Integer(1))
y
>>> O.gen(Integer(2))
(1/(x^3 + x^2 + x))*y^2
>>> O.gen(Integer(3))
Traceback (most recent call last):

```

(continues on next page)

(continued from previous page)

```
...
IndexError: there are only 3 generators
```

ideal(*gens, **kwargs)

Return the fractional ideal generated by the elements in gens.

INPUT:

- gens – list of generators

EXAMPLES:

```
sage: # needs sage.rings.finite_rings
sage: K.<x> = FunctionField(GF(7)); R.<y> = K[]
sage: O = K.maximal_order()
sage: I = O.ideal(x^2 - 4)
sage: L.<y> = K.extension(y^2 - x^3 - 1)
sage: S = L.maximal_order()
sage: S.ideal(1/y)
Ideal ((1/(x^3 + 1))*y) of Maximal order of Function field
in y defined by y^2 + 6*x^3 + 6
sage: I2 = S.ideal(x^2 - 4); I2
Ideal (x^2 + 3) of Maximal order of Function field in y defined by y^2 + 6*x^
↪ 3 + 6
sage: I2 == S.ideal(I)
True

sage: K.<x> = FunctionField(QQ); R.<y> = K[]
sage: O = K.maximal_order()
sage: I = O.ideal(x^2 - 4)
sage: L.<y> = K.extension(y^2 - x^3 - 1)
sage: S = L.maximal_order()
sage: S.ideal(1/y)
Ideal ((1/(x^3 + 1))*y) of
Maximal order of Function field in y defined by y^2 - x^3 - 1
sage: I2 = S.ideal(x^2-4); I2
Ideal (x^2 - 4) of Maximal order of Function field in y defined by y^2 - x^3 -
↪ 1
sage: I2 == S.ideal(I)
True
```

```
>>> from sage.all import *
>>> # needs sage.rings.finite_rings
>>> K = FunctionField(GF(Integer(7)), names=('x',)); (x,) = K._first_ngens(1);
↪ R = K['y']; (y,) = R._first_ngens(1)
>>> O = K.maximal_order()
>>> I = O.ideal(x**Integer(2) - Integer(4))
>>> L = K.extension(y**Integer(2) - x**Integer(3) - Integer(1), names=('y',));
↪ (y,) = L._first_ngens(1)
>>> S = L.maximal_order()
>>> S.ideal(Integer(1)/y)
Ideal ((1/(x^3 + 1))*y) of Maximal order of Function field
in y defined by y^2 + 6*x^3 + 6
```

(continues on next page)

(continued from previous page)

```

>>> I2 = S.ideal(x**Integer(2) - Integer(4)); I2
Ideal (x^2 + 3) of Maximal order of Function field in y defined by y^2 + 6*x^
↪ 3 + 6
>>> I2 == S.ideal(I)
True

>>> K = FunctionField(QQ, names=('x',)); (x,) = K._first_ngens(1); R = K['y'];
↪ (y,) = R._first_ngens(1)
>>> O = K.maximal_order()
>>> I = O.ideal(x**Integer(2) - Integer(4))
>>> L = K.extension(y**Integer(2) - x**Integer(3) - Integer(1), names=('y',));
↪ (y,) = L._first_ngens(1)
>>> S = L.maximal_order()
>>> S.ideal(Integer(1)/y)
Ideal ((1/(x^3 + 1))*y) of
Maximal order of Function field in y defined by y^2 - x^3 - 1
>>> I2 = S.ideal(x**Integer(2)-Integer(4)); I2
Ideal (x^2 - 4) of Maximal order of Function field in y defined by y^2 - x^3 -
↪ 1
>>> I2 == S.ideal(I)
True

```

ideal_with_gens_over_base (*gens*)

Return the fractional ideal with basis *gens* over the maximal order of the base field.

INPUT:

- *gens* – list of elements that generates the ideal over the maximal order of the base field

EXAMPLES:

```

sage: # needs sage.rings.finite_rings
sage: K.<x> = FunctionField(GF(7)); R.<y> = K[]
sage: L.<y> = K.extension(y^2 - x^3 - 1)
sage: O = L.maximal_order(); O
Maximal order of Function field in y defined by y^2 + 6*x^3 + 6
sage: I = O.ideal_with_gens_over_base([1, y]); I
Ideal (1) of Maximal order of Function field in y defined by y^2 + 6*x^3 + 6
sage: I.module()
Free module of degree 2 and rank 2 over
Maximal order of Rational function field in x over Finite Field of size 7
Echelon basis matrix:
[1 0]
[0 1]

```

```

>>> from sage.all import *
>>> # needs sage.rings.finite_rings
>>> K = FunctionField(GF(Integer(7)), names=('x',)); (x,) = K._first_ngens(1);
↪ R = K['y']; (y,) = R._first_ngens(1)
>>> L = K.extension(y**Integer(2) - x**Integer(3) - Integer(1), names=('y',));
↪ (y,) = L._first_ngens(1)
>>> O = L.maximal_order(); O
Maximal order of Function field in y defined by y^2 + 6*x^3 + 6

```

(continues on next page)

(continued from previous page)

```
>>> I = O.ideal_with_gens_over_base([Integer(1), y]); I
Ideal (1) of Maximal order of Function field in y defined by y^2 + 6*x^3 + 6
>>> I.module()
Free module of degree 2 and rank 2 over
Maximal order of Rational function field in x over Finite Field of size 7
Echelon basis matrix:
[1 0]
[0 1]
```

There is no check if the resulting object is really an ideal:

```
sage: # needs sage.rings.finite_rings
sage: K.<x> = FunctionField(GF(7)); R.<y> = K[]
sage: L.<y> = K.extension(y^2 - x^3 - 1)
sage: O = L.equation_order()
sage: I = O.ideal_with_gens_over_base([y]); I
Ideal (y) of Order in Function field in y defined by y^2 + 6*x^3 + 6
sage: y in I
True
sage: y^2 in I
False
```

```
>>> from sage.all import *
>>> # needs sage.rings.finite_rings
>>> K = FunctionField(GF(Integer(7)), names=('x',)); (x,) = K._first_ngens(1);
↪ R = K['y']; (y,) = R._first_ngens(1)
>>> L = K.extension(y**Integer(2) - x**Integer(3) - Integer(1), names=('y',));
↪ (y,) = L._first_ngens(1)
>>> O = L.equation_order()
>>> I = O.ideal_with_gens_over_base([y]); I
Ideal (y) of Order in Function field in y defined by y^2 + 6*x^3 + 6
>>> y in I
True
>>> y**Integer(2) in I
False
```

ngens()

Return the number of generators of the order.

EXAMPLES:

```
sage: # needs sage.rings.finite_rings
sage: K.<x> = FunctionField(GF(2)); _.<t> = K[]
sage: L.<y> = K.extension(t^3 - x^2*(x^2 + x + 1)^2)
sage: Oinf = L.maximal_order()
sage: Oinf.ngens()
3
```

```
>>> from sage.all import *
>>> # needs sage.rings.finite_rings
>>> K = FunctionField(GF(Integer(2)), names=('x',)); (x,) = K._first_ngens(1);
↪ _ = K['t']; (t,) = _._first_ngens(1)
```

(continues on next page)

(continued from previous page)

```

>>> L = K.extension(t**Integer(3) - x**Integer(2)*(x**Integer(2) + x +
↳ Integer(1))**Integer(2), names=('y',)); (y,) = L._first_ngens(1)
>>> Oinf = L.maximal_order()
>>> Oinf.ngens()
3

```

polynomial()

Return the defining polynomial of the function field of which this is an order.

EXAMPLES:

```

sage: # needs sage.rings.finite_rings
sage: K.<x> = FunctionField(GF(7)); R.<y> = K[]
sage: L.<y> = K.extension(y^4 + x*y + 4*x + 1)
sage: O = L.equation_order()
sage: O.polynomial()
y^4 + x*y + 4*x + 1

sage: K.<x> = FunctionField(QQ); R.<y> = K[]
sage: L.<y> = K.extension(y^4 + x*y + 4*x + 1)
sage: O = L.equation_order()
sage: O.polynomial()
y^4 + x*y + 4*x + 1

```

```

>>> from sage.all import *
>>> # needs sage.rings.finite_rings
>>> K = FunctionField(GF(Integer(7)), names=('x',)); (x,) = K._first_ngens(1);
↳ R = K['y']; (y,) = R._first_ngens(1)
>>> L = K.extension(y**Integer(4) + x*y + Integer(4)*x + Integer(1), names=('y
↳ ',)); (y,) = L._first_ngens(1)
>>> O = L.equation_order()
>>> O.polynomial()
y^4 + x*y + 4*x + 1

>>> K = FunctionField(QQ, names=('x',)); (x,) = K._first_ngens(1); R = K['y'];
↳ (y,) = R._first_ngens(1)
>>> L = K.extension(y**Integer(4) + x*y + Integer(4)*x + Integer(1), names=('y
↳ ',)); (y,) = L._first_ngens(1)
>>> O = L.equation_order()
>>> O.polynomial()
y^4 + x*y + 4*x + 1

```

IDEALS OF FUNCTION FIELDS

Ideals of an order of a function field include all fractional ideals of the order. Sage provides basic arithmetic with fractional ideals.

The fractional ideals of the maximal order of a global function field forms a multiplicative monoid. Sage allows advanced arithmetic with the fractional ideals. For example, an ideal of the maximal order can be factored into a product of prime ideals.

EXAMPLES:

Ideals in the maximal order of a rational function field:

```
sage: K.<x> = FunctionField(QQ)
sage: O = K.maximal_order()
sage: I = O.ideal(x^3 + 1); I
Ideal (x^3 + 1) of Maximal order of Rational function field in x over Rational Field
sage: I^2
Ideal (x^6 + 2*x^3 + 1) of Maximal order of Rational function field in x over
↳Rational Field
sage: ~I
Ideal (1/(x^3 + 1)) of Maximal order of Rational function field in x over Rational
↳Field
sage: ~I * I
Ideal (1) of Maximal order of Rational function field in x over Rational Field
```

```
>>> from sage.all import *
>>> K = FunctionField(QQ, names=('x',)); (x,) = K._first_ngens(1)
>>> O = K.maximal_order()
>>> I = O.ideal(x**Integer(3) + Integer(1)); I
Ideal (x^3 + 1) of Maximal order of Rational function field in x over Rational Field
>>> I**Integer(2)
Ideal (x^6 + 2*x^3 + 1) of Maximal order of Rational function field in x over
↳Rational Field
>>> ~I
Ideal (1/(x^3 + 1)) of Maximal order of Rational function field in x over Rational
↳Field
>>> ~I * I
Ideal (1) of Maximal order of Rational function field in x over Rational Field
```

Ideals in the equation order of an extension of a rational function field:

```
sage: K.<x> = FunctionField(QQ); R.<y> = K[]
sage: L.<y> = K.extension(y^2 - x^3 - 1)
```

(continues on next page)

(continued from previous page)

```

→                                     # needs sage.rings.function_field
sage: O = L.equation_order()
→                                     # needs sage.rings.function_field
sage: I = O.ideal(y); I
→                                     # needs sage.rings.function_field
Ideal (x^3 + 1, -y) of Order in Function field in y defined by y^2 - x^3 - 1
sage: I^2
→                                     # needs sage.rings.function_field
Ideal (x^3 + 1, (-x^3 - 1)*y) of Order in Function field in y defined by y^2 - x^3 - 1

```

```

>>> from sage.all import *
>>> K = FunctionField(QQ, names=('x',)); (x,) = K._first_ngens(1); R = K['y']; (y,) =
→ R._first_ngens(1)
>>> L = K.extension(y**Integer(2) - x**Integer(3) - Integer(1), names=('y',)); (y,) =
→ L._first_ngens(1) # needs sage.rings.function_field
>>> O = L.equation_order()
→                                     # needs sage.rings.function_field
>>> I = O.ideal(y); I
→                                     # needs sage.rings.function_field
Ideal (x^3 + 1, -y) of Order in Function field in y defined by y^2 - x^3 - 1
>>> I**Integer(2)
→                                     # needs sage.rings.function_field
Ideal (x^3 + 1, (-x^3 - 1)*y) of Order in Function field in y defined by y^2 - x^3 - 1

```

Ideals in the maximal order of a global function field:

```

sage: # needs sage.rings.finite_rings
sage: K.<x> = FunctionField(GF(2)); R.<y> = K[]
sage: L.<y> = K.extension(y^2 - x^3*y - x)
→                                     # needs sage.rings.function_field
sage: O = L.maximal_order()
→                                     # needs sage.rings.function_field
sage: I = O.ideal(y)
→                                     # needs sage.rings.function_field
sage: I^2
→                                     # needs sage.rings.function_field
Ideal (x) of Maximal order of Function field in y defined by y^2 + x^3*y + x
sage: ~I
→                                     # needs sage.rings.function_field
Ideal (1/x*y) of Maximal order of Function field in y defined by y^2 + x^3*y + x
sage: ~I * I
→                                     # needs sage.rings.function_field
Ideal (1) of Maximal order of Function field in y defined by y^2 + x^3*y + x

sage: J = O.ideal(x + y) * I
→                                     # needs sage.rings.finite_rings sage.rings.function_
→ field
sage: J.factor()
→                                     # needs sage.rings.finite_rings sage.rings.function_
→ field
(Ideal (y) of Maximal order of Function field in y defined by y^2 + x^3*y + x)^2 *
(Ideal (x^3 + x + 1, y + x) of Maximal order of Function field in y defined by y^2 +

```

(continues on next page)

(continued from previous page)

```
↪ x^3*y + x)
```

```
>>> from sage.all import *
>>> # needs sage.rings.finite_rings
>>> K = FunctionField(GF(Integer(2)), names=('x',)); (x,) = K._first_ngens(1); R = K[
↪ 'y']; (y,) = R._first_ngens(1)
>>> L = K.extension(y**Integer(2) - x**Integer(3)*y - x, names=('y',)); (y,) = L._
↪ first_ngens(1) # needs sage.rings.function_field
>>> O = L.maximal_order()
↪ # needs sage.rings.function_field
>>> I = O.ideal(y)
↪ # needs sage.rings.function_field
>>> I**Integer(2)
↪ # needs sage.rings.function_field
Ideal (x) of Maximal order of Function field in y defined by y^2 + x^3*y + x
>>> ~I
↪ # needs sage.rings.function_field
Ideal (1/x*y) of Maximal order of Function field in y defined by y^2 + x^3*y + x
>>> ~I * I
↪ # needs sage.rings.function_field
Ideal (1) of Maximal order of Function field in y defined by y^2 + x^3*y + x

>>> J = O.ideal(x + y) * I
↪ # needs sage.rings.finite_rings sage.rings.function_
↪ field
>>> J.factor()
↪ # needs sage.rings.finite_rings sage.rings.function_
↪ field
(Ideal (y) of Maximal order of Function field in y defined by y^2 + x^3*y + x)^2 *
(Ideal (x^3 + x + 1, y + x) of Maximal order of Function field in y defined by y^2 +
↪ x^3*y + x)
```

Ideals in the maximal infinite order of a global function field:

```
sage: # needs sage.rings.finite_rings
sage: K.<x> = FunctionField(GF(3^2)); R.<t> = K[]
sage: F.<y> = K.extension(t^3 + t^2 - x^4)
↪ # needs sage.rings.function_field
sage: Oinf = F.maximal_order_infinite()
↪ # needs sage.rings.function_field
sage: I = Oinf.ideal(1/y)
↪ # needs sage.rings.function_field
sage: I + I == I
True
sage: I^2
↪ # needs sage.rings.function_field
Ideal (1/x^4*y) of Maximal infinite order of Function field in y defined by y^3 + y^2
↪ + 2*x^4
sage: ~I
↪ # needs sage.rings.function_field
Ideal (y) of Maximal infinite order of Function field in y defined by y^3 + y^2 + 2*x^
↪ 4
```

(continues on next page)

(continued from previous page)

```

sage: ~I * I
↪ # needs sage.rings.function_field
Ideal (1) of Maximal infinite order of Function field in y defined by y^3 + y^2 + 2*x^
↪ 4
sage: I.factor()
↪ # needs sage.rings.function_field
(Ideal (1/x^3*y^2) of Maximal infinite order of Function field in y defined by y^3 +
↪ y^2 + 2*x^4)^4

```

```

>>> from sage.all import *
>>> # needs sage.rings.finite_rings
>>> K = FunctionField(GF(Integer(3)**Integer(2)), names=('x',)); (x,) = K._first_
↪ ngens(1); R = K['t']; (t,) = R._first_ngens(1)
>>> F = K.extension(t**Integer(3) + t**Integer(2) - x**Integer(4), names=('y',)); (y,
↪ ) = F._first_ngens(1) # needs sage.rings.function_field
>>> Oinf = F.maximal_order_infinite()
↪ # needs sage.rings.function_field
>>> I = Oinf.ideal(Integer(1)/y)
↪ # needs sage.rings.function_field
>>> I + I == I
True
>>> I**Integer(2)
↪ # needs sage.rings.function_field
Ideal (1/x^4*y) of Maximal infinite order of Function field in y defined by y^3 + y^2
↪ + 2*x^4
>>> ~I
↪ # needs sage.rings.function_field
Ideal (y) of Maximal infinite order of Function field in y defined by y^3 + y^2 + 2*x^
↪ 4
>>> ~I * I
↪ # needs sage.rings.function_field
Ideal (1) of Maximal infinite order of Function field in y defined by y^3 + y^2 + 2*x^
↪ 4
>>> I.factor()
↪ # needs sage.rings.function_field
(Ideal (1/x^3*y^2) of Maximal infinite order of Function field in y defined by y^3 +
↪ y^2 + 2*x^4)^4

```

AUTHORS:

- William Stein (2010): initial version
- Maarten Derickx (2011-09-14): fixed `ideal_with_gens_over_base()`
- Kwankyu Lee (2017-04-30): added ideals for global function fields

class sage.rings.function_field.ideal.**FunctionFieldIdeal** (ring)

Bases: `Element`

Base class of fractional ideals of function fields.

INPUT:

- ring – ring of the ideal

EXAMPLES:

```

sage: K.<x> = FunctionField(GF(7))
sage: O = K.equation_order()
sage: O.ideal(x^3 + 1)
Ideal (x^3 + 1) of Maximal order of Rational function field in x over Finite
↪Field of size 7

```

```

>>> from sage.all import *
>>> K = FunctionField(GF(Integer(7)), names=('x',)); (x,) = K._first_ngens(1)
>>> O = K.equation_order()
>>> O.ideal(x**Integer(3) + Integer(1))
Ideal (x^3 + 1) of Maximal order of Rational function field in x over Finite
↪Field of size 7

```

base_ring()

Return the base ring of this ideal.

EXAMPLES:

```

sage: # needs sage.rings.function_field
sage: K.<x> = FunctionField(QQ); R.<y> = K[]
sage: L.<y> = K.extension(y^2 - x^3 - 1)
sage: O = L.equation_order()
sage: I = O.ideal(x^2 + 1)
sage: I.base_ring()
Order in Function field in y defined by y^2 - x^3 - 1

```

```

>>> from sage.all import *
>>> # needs sage.rings.function_field
>>> K = FunctionField(QQ, names=('x',)); (x,) = K._first_ngens(1); R = K['y'];
↪ (y,) = R._first_ngens(1)
>>> L = K.extension(y**Integer(2) - x**Integer(3) - Integer(1), names=('y',));
↪ (y,) = L._first_ngens(1)
>>> O = L.equation_order()
>>> I = O.ideal(x**Integer(2) + Integer(1))
>>> I.base_ring()
Order in Function field in y defined by y^2 - x^3 - 1

```

divisor()

Return the divisor corresponding to the ideal.

EXAMPLES:

```

sage: # needs sage.modules sage.rings.finite_rings
sage: K.<x> = FunctionField(GF(4))
sage: O = K.maximal_order()
sage: I = O.ideal(x*(x + 1)^2/(x^2 + x + 1))
sage: I.divisor()
Place (x) + 2*Place (x + 1) - Place (x + z2) - Place (x + z2 + 1)

sage: # needs sage.modules sage.rings.finite_rings
sage: Oinf = K.maximal_order_infinite()
sage: I = Oinf.ideal((x + 1)/(x^3 + 1))
sage: I.divisor()

```

(continues on next page)

(continued from previous page)

```

2*Place (1/x)

sage: # needs sage.rings.function_field
sage: K.<x> = FunctionField(GF(2)); _.<T> = PolynomialRing(K)
sage: F.<y> = K.extension(T^3 - x^2*(x^2 + x + 1)^2)
sage: O = F.maximal_order()
sage: I = O.ideal(y)
sage: I.divisor()
2*Place (x, (1/(x^3 + x^2 + x))*y^2)
+ 2*Place (x^2 + x + 1, (1/(x^3 + x^2 + x))*y^2)

sage: # needs sage.rings.function_field
sage: Oinf = F.maximal_order_infinite()
sage: I = Oinf.ideal(y)
sage: I.divisor()
-2*Place (1/x, 1/x^4*y^2 + 1/x^2*y + 1)
- 2*Place (1/x, 1/x^2*y + 1)

sage: # needs sage.rings.function_field
sage: K.<x> = FunctionField(GF(2)); _.<Y> = K[]
sage: L.<y> = K.extension(Y^2 + Y + x + 1/x)
sage: O = L.maximal_order()
sage: I = O.ideal(y)
sage: I.divisor()
- Place (x, x*y)
+ 2*Place (x + 1, x*y)

sage: # needs sage.rings.function_field
sage: Oinf = L.maximal_order_infinite()
sage: I = Oinf.ideal(y)
sage: I.divisor()
- Place (1/x, 1/x*y)

```

```

>>> from sage.all import *
>>> # needs sage.modules sage.rings.finite_rings
>>> K = FunctionField(GF(Integer(4)), names=('x',)); (x,) = K._first_ngens(1)
>>> O = K.maximal_order()
>>> I = O.ideal(x*(x + Integer(1))**Integer(2)/(x**Integer(2) + x +
↳ Integer(1)))
>>> I.divisor()
Place (x) + 2*Place (x + 1) - Place (x + z2) - Place (x + z2 + 1)

>>> # needs sage.modules sage.rings.finite_rings
>>> Oinf = K.maximal_order_infinite()
>>> I = Oinf.ideal((x + Integer(1))/(x**Integer(3) + Integer(1)))
>>> I.divisor()
2*Place (1/x)

>>> # needs sage.rings.function_field
>>> K = FunctionField(GF(Integer(2)), names=('x',)); (x,) = K._first_ngens(1);
↳ _ = PolynomialRing(K, names=('T',)); (T,) = _.first_ngens(1)
>>> F = K.extension(T**Integer(3) - x**Integer(2)*(x**Integer(2) + x +
↳

```

(continues on next page)

(continued from previous page)

```

↪ Integer(1)**Integer(2), names=('y',)); (y,) = F._first_ngens(1)
>>> O = F.maximal_order()
>>> I = O.ideal(y)
>>> I.divisor()
2*Place (x, (1/(x^3 + x^2 + x))*y^2)
+ 2*Place (x^2 + x + 1, (1/(x^3 + x^2 + x))*y^2)

>>> # needs sage.rings.function_field
>>> Oinf = F.maximal_order_infinite()
>>> I = Oinf.ideal(y)
>>> I.divisor()
-2*Place (1/x, 1/x^4*y^2 + 1/x^2*y + 1)
- 2*Place (1/x, 1/x^2*y + 1)

>>> # needs sage.rings.function_field
>>> K = FunctionField(GF(Integer(2)), names=('x',)); (x,) = K._first_ngens(1);
↪ _ = K['Y']; (Y,) = _._first_ngens(1)
>>> L = K.extension(Y**Integer(2) + Y + x + Integer(1)/x, names=('y',)); (y,) ↪
↪ = L._first_ngens(1)
>>> O = L.maximal_order()
>>> I = O.ideal(y)
>>> I.divisor()
- Place (x, x*y)
+ 2*Place (x + 1, x*y)

>>> # needs sage.rings.function_field
>>> Oinf = L.maximal_order_infinite()
>>> I = Oinf.ideal(y)
>>> I.divisor()
- Place (1/x, 1/x*y)

```

divisor_of_poles()

Return the divisor of poles corresponding to the ideal.

EXAMPLES:

```

sage: # needs sage.modules sage.rings.finite_rings
sage: K.<x> = FunctionField(GF(4))
sage: O = K.maximal_order()
sage: I = O.ideal(x*(x + 1)^2/(x^2 + x + 1))
sage: I.divisor_of_poles()
Place (x + z2) + Place (x + z2 + 1)

sage: # needs sage.modules
sage: K.<x> = FunctionField(GF(2))
sage: Oinf = K.maximal_order_infinite()
sage: I = Oinf.ideal((x + 1)/(x^3 + 1))
sage: I.divisor_of_poles()
0

sage: # needs sage.rings.function_field
sage: K.<x> = FunctionField(GF(2)); _.<Y> = K[]

```

(continues on next page)

(continued from previous page)

```
sage: L.<y> = K.extension(Y^2 + Y + x + 1/x)
sage: O = L.maximal_order()
sage: I = O.ideal(y)
sage: I.divisor_of_poles()
Place (x, x*y)
```

```
>>> from sage.all import *
>>> # needs sage.modules sage.rings.finite_rings
>>> K = FunctionField(GF(Integer(4)), names=('x',)); (x,) = K._first_ngens(1)
>>> O = K.maximal_order()
>>> I = O.ideal(x*(x + Integer(1))*Integer(2)/(x**Integer(2) + x +
↳ Integer(1)))
>>> I.divisor_of_poles()
Place (x + z2) + Place (x + z2 + 1)

>>> # needs sage.modules
>>> K = FunctionField(GF(Integer(2)), names=('x',)); (x,) = K._first_ngens(1)
>>> Oinf = K.maximal_order_infinite()
>>> I = Oinf.ideal((x + Integer(1))/(x**Integer(3) + Integer(1)))
>>> I.divisor_of_poles()
0

>>> # needs sage.rings.function_field
>>> K = FunctionField(GF(Integer(2)), names=('x',)); (x,) = K._first_ngens(1);
↳ _ = K['Y']; (Y,) = _._first_ngens(1)
>>> L = K.extension(Y**Integer(2) + Y + x + Integer(1)/x, names=('y',)); (y,) ↳
↳ L._first_ngens(1)
>>> O = L.maximal_order()
>>> I = O.ideal(y)
>>> I.divisor_of_poles()
Place (x, x*y)
```

divisor_of_zeros()

Return the divisor of zeros corresponding to the ideal.

EXAMPLES:

```
sage: # needs sage.modules sage.rings.finite_rings
sage: K.<x> = FunctionField(GF(4))
sage: O = K.maximal_order()
sage: I = O.ideal(x*(x + 1)^2/(x^2 + x + 1))
sage: I.divisor_of_zeros()
Place (x) + 2*Place (x + 1)

sage: # needs sage.modules
sage: K.<x> = FunctionField(GF(2))
sage: Oinf = K.maximal_order_infinite()
sage: I = Oinf.ideal((x + 1)/(x^3 + 1))
sage: I.divisor_of_zeros()
2*Place (1/x)

sage: # needs sage.rings.function_field
```

(continues on next page)

(continued from previous page)

```

sage: K.<x> = FunctionField(GF(2)); _.<Y> = K[]
sage: L.<y> = K.extension(Y^2 + Y + x + 1/x)
sage: O = L.maximal_order()
sage: I = O.ideal(y)
sage: I.divisor_of_zeros()
2*Place (x + 1, x*y)

```

```

>>> from sage.all import *
>>> # needs sage.modules sage.rings.finite_rings
>>> K = FunctionField(GF(Integer(4)), names=('x',)); (x,) = K._first_ngens(1)
>>> O = K.maximal_order()
>>> I = O.ideal(x*(x + Integer(1))**Integer(2)/(x**Integer(2) + x +
↳ Integer(1)))
>>> I.divisor_of_zeros()
Place (x) + 2*Place (x + 1)

>>> # needs sage.modules
>>> K = FunctionField(GF(Integer(2)), names=('x',)); (x,) = K._first_ngens(1)
>>> Oinf = K.maximal_order_infinite()
>>> I = Oinf.ideal((x + Integer(1))/(x**Integer(3) + Integer(1)))
>>> I.divisor_of_zeros()
2*Place (1/x)

>>> # needs sage.rings.function_field
>>> K = FunctionField(GF(Integer(2)), names=('x',)); (x,) = K._first_ngens(1);
↳ _ = K['Y']; (Y,) = _.first_ngens(1)
>>> L = K.extension(Y**Integer(2) + Y + x + Integer(1)/x, names=('y',)); (y,) ↳
↳ L._first_ngens(1)
>>> O = L.maximal_order()
>>> I = O.ideal(y)
>>> I.divisor_of_zeros()
2*Place (x + 1, x*y)

```

factor()

Return the factorization of this ideal.

Subclass of this class should define `_factor()` method that returns a list of prime ideal and multiplicity pairs.

EXAMPLES:

```

sage: # needs sage.rings.finite_rings
sage: K.<x> = FunctionField(GF(4))
sage: O = K.maximal_order()
sage: I = O.ideal(x^3*(x + 1)^2)
sage: I.factor()
(Ideal (x) of Maximal order of Rational function field in x
over Finite Field in z2 of size 2^2)^3 *
(Ideal (x + 1) of Maximal order of Rational function field in x
over Finite Field in z2 of size 2^2)^2

sage: # needs sage.rings.finite_rings
sage: Oinf = K.maximal_order_infinite()

```

(continues on next page)

(continued from previous page)

```

sage: I = Oinf.ideal((x + 1)/(x^3 + 1))
sage: I.factor()
(Ideal (1/x) of Maximal infinite order of Rational function field in x
over Finite Field in z2 of size 2^2)^2

sage: # needs sage.rings.function_field
sage: K.<x> = FunctionField(GF(2)); _.<T> = PolynomialRing(K)
sage: F.<y> = K.extension(T^3 - x^2*(x^2 + x + 1)^2)
sage: O = F.maximal_order()
sage: I = O.ideal(y)
sage: I == I.factor().prod()
True

sage: # needs sage.rings.function_field
sage: Oinf = F.maximal_order_infinite()
sage: f = 1/x
sage: I = Oinf.ideal(f)
sage: I.factor()
(Ideal ((1/(x^4 + x^3 + x^2))*y^2 + 1/x^2*y + 1) of Maximal infinite order
of Function field in y defined by y^3 + x^6 + x^4 + x^2) *
(Ideal ((1/(x^4 + x^3 + x^2))*y^2 + 1) of Maximal infinite order
of Function field in y defined by y^3 + x^6 + x^4 + x^2)

sage: # needs sage.rings.function_field
sage: K.<x> = FunctionField(QQ); _.<Y> = K[]
sage: F.<y> = K.extension(Y^3 - x^2*(x^2 + x + 1)^2)
sage: O = F.maximal_order()
sage: I = O.ideal(y)
sage: I == I.factor().prod()
True

sage: # needs sage.rings.function_field
sage: K.<x> = FunctionField(QQ); _.<Y> = K[]
sage: L.<y> = K.extension(Y^2 + Y + x + 1/x)
sage: O = L.maximal_order()
sage: I = O.ideal(y)
sage: I == I.factor().prod()
True

```

```

>>> from sage.all import *
>>> # needs sage.rings.finite_rings
>>> K = FunctionField(GF(Integer(4)), names=('x',)); (x,) = K._first_ngens(1)
>>> O = K.maximal_order()
>>> I = O.ideal(x**Integer(3)*(x + Integer(1))**Integer(2))
>>> I.factor()
(Ideal (x) of Maximal order of Rational function field in x
over Finite Field in z2 of size 2^2)^3 *
(Ideal (x + 1) of Maximal order of Rational function field in x
over Finite Field in z2 of size 2^2)^2

>>> # needs sage.rings.finite_rings
>>> Oinf = K.maximal_order_infinite()

```

(continues on next page)

(continued from previous page)

```

>>> I = Oinf.ideal((x + Integer(1))/(x**Integer(3) + Integer(1)))
>>> I.factor()
(Ideal (1/x) of Maximal infinite order of Rational function field in x
over Finite Field in z2 of size 2^2)^2

>>> # needs sage.rings.function_field
>>> K = FunctionField(GF(Integer(2)), names=('x',)); (x,) = K._first_ngens(1);
↪ _ = PolynomialRing(K, names=('T',)); (T,) = _._first_ngens(1)
>>> F = K.extension(T**Integer(3) - x**Integer(2)*(x**Integer(2) + x +
↪ Integer(1))**Integer(2), names=('y',)); (y,) = F._first_ngens(1)
>>> O = F.maximal_order()
>>> I = O.ideal(y)
>>> I == I.factor().prod()
True

>>> # needs sage.rings.function_field
>>> Oinf = F.maximal_order_infinite()
>>> f = Integer(1)/x
>>> I = Oinf.ideal(f)
>>> I.factor()
(Ideal ((1/(x^4 + x^3 + x^2))*y^2 + 1/x^2*y + 1) of Maximal infinite order
of Function field in y defined by y^3 + x^6 + x^4 + x^2) *
(Ideal ((1/(x^4 + x^3 + x^2))*y^2 + 1) of Maximal infinite order
of Function field in y defined by y^3 + x^6 + x^4 + x^2)

>>> # needs sage.rings.function_field
>>> K = FunctionField(QQ, names=('x',)); (x,) = K._first_ngens(1); _ = K['Y'];
↪ (Y,) = _._first_ngens(1)
>>> F = K.extension(Y**Integer(3) - x**Integer(2)*(x**Integer(2) + x +
↪ Integer(1))**Integer(2), names=('y',)); (y,) = F._first_ngens(1)
>>> O = F.maximal_order()
>>> I = O.ideal(y)
>>> I == I.factor().prod()
True

>>> # needs sage.rings.function_field
>>> K = FunctionField(QQ, names=('x',)); (x,) = K._first_ngens(1); _ = K['Y'];
↪ (Y,) = _._first_ngens(1)
>>> L = K.extension(Y**Integer(2) + Y + x + Integer(1)/x, names=('y',)); (y,)
↪ L._first_ngens(1)
>>> O = L.maximal_order()
>>> I = O.ideal(y)
>>> I == I.factor().prod()
True

```

gens_reduced()

Return reduced generators.

For now, this method just looks at the generators and sees if any can be removed without changing the ideal. It prefers principal representations (a single generator) over all others, and otherwise picks the generator set with the shortest print representation.

This method is provided so that ideals in function fields have the method `gens_reduced()`, just like ideals

of number fields. Sage linear algebra machinery sometimes requires this.

EXAMPLES:

```
sage: K.<x> = FunctionField(GF(7))
sage: O = K.equation_order()
sage: I = O.ideal(x, x^2, x^2 + x)
sage: I.gens_reduced()
(x,)
```

```
>>> from sage.all import *
>>> K = FunctionField(GF(Integer(7)), names=('x',)); (x,) = K._first_ngens(1)
>>> O = K.equation_order()
>>> I = O.ideal(x, x**Integer(2), x**Integer(2) + x)
>>> I.gens_reduced()
(x,)
```

place()

Return the place associated with this prime ideal.

EXAMPLES:

```
sage: # needs sage.rings.finite_rings
sage: K.<x> = FunctionField(GF(4))
sage: O = K.maximal_order()
sage: I = O.ideal(x^2 + x + 1)
sage: I.place()
Traceback (most recent call last):
...
TypeError: not a prime ideal
sage: I = O.ideal(x^3 + x + 1)
sage: I.place()
Place (x^3 + x + 1)

sage: K.<x> = FunctionField(GF(2))
sage: Oinf = K.maximal_order_infinite()
sage: I = Oinf.ideal((x + 1)/(x^3 + 1))
sage: p = I.factor()[0][0]
sage: p.place()
Place (1/x)

sage: # needs sage.rings.function_field
sage: K.<x> = FunctionField(GF(2)); _.<t> = PolynomialRing(K)
sage: F.<y> = K.extension(t^3 - x^2*(x^2+x+1)^2)
sage: O = F.maximal_order()
sage: I = O.ideal(y)
sage: [f.place() for f,_ in I.factor()]
[Place (x, (1/(x^3 + x^2 + x))*y^2),
 Place (x^2 + x + 1, (1/(x^3 + x^2 + x))*y^2)]

sage: # needs sage.rings.function_field
sage: K.<x> = FunctionField(GF(2)); _.<Y> = K[]
sage: L.<y> = K.extension(Y^2 + Y + x + 1/x)
sage: O = L.maximal_order()
```

(continues on next page)

(continued from previous page)

```

sage: I = O.ideal(y)
sage: [f.place() for f,_ in I.factor()]
[Place (x, x*y), Place (x + 1, x*y)]

sage: # needs sage.rings.finite_rings sage.rings.function_field
sage: K.<x> = FunctionField(GF(3^2)); R.<t> = PolynomialRing(K)
sage: F.<y> = K.extension(t^3 + t^2 - x^4)
sage: Oinf = F.maximal_order_infinite()
sage: I = Oinf.ideal(1/x)
sage: I.factor()
(Ideal (1/x^3*y^2) of Maximal infinite order of Function field
in y defined by y^3 + y^2 + 2*x^4)^3
sage: J = I.factor()[0][0]
sage: J.is_prime()
True
sage: J.place()
Place (1/x, 1/x^3*y^2)

sage: # needs sage.rings.function_field
sage: K.<x> = FunctionField(GF(2)); L.<Y> = K[]
sage: L.<y> = K.extension(Y^2 + Y + x + 1/x)
sage: Oinf = L.maximal_order_infinite()
sage: I = Oinf.ideal(1/x)
sage: I.factor()
(Ideal (1/x*y) of Maximal infinite order of Function field in y
defined by y^2 + y + (x^2 + 1)/x)^2
sage: J = I.factor()[0][0]
sage: J.is_prime()
True
sage: J.place()
Place (1/x, 1/x*y)

```

```

>>> from sage.all import *
>>> # needs sage.rings.finite_rings
>>> K = FunctionField(GF(Integer(4)), names=('x',)); (x,) = K._first_ngens(1)
>>> O = K.maximal_order()
>>> I = O.ideal(x**Integer(2) + x + Integer(1))
>>> I.place()
Traceback (most recent call last):
...
TypeError: not a prime ideal
>>> I = O.ideal(x**Integer(3) + x + Integer(1))
>>> I.place()
Place (x^3 + x + 1)

>>> K = FunctionField(GF(Integer(2)), names=('x',)); (x,) = K._first_ngens(1)
>>> Oinf = K.maximal_order_infinite()
>>> I = Oinf.ideal((x + Integer(1))/(x**Integer(3) + Integer(1)))
>>> p = I.factor()[Integer(0)][Integer(0)]
>>> p.place()
Place (1/x)

```

(continues on next page)

(continued from previous page)

```

>>> # needs sage.rings.function_field
>>> K = FunctionField(GF(Integer(2)), names=('x',)); (x,) = K._first_ngens(1);
↳ _ = PolynomialRing(K, names=('t',)); (t,) = _._first_ngens(1)
>>> F = K.extension(t**Integer(3) -
↳ x**Integer(2)*(x**Integer(2)+x+Integer(1))**Integer(2), names=('y',)); (y,) =
↳ F._first_ngens(1)
>>> O = F.maximal_order()
>>> I = O.ideal(y)
>>> [f.place() for f,_ in I.factor()]
[Place (x, (1/(x^3 + x^2 + x))*y^2),
 Place (x^2 + x + 1, (1/(x^3 + x^2 + x))*y^2)]

>>> # needs sage.rings.function_field
>>> K = FunctionField(GF(Integer(2)), names=('x',)); (x,) = K._first_ngens(1);
↳ _ = K['Y']; (Y,) = _._first_ngens(1)
>>> L = K.extension(Y**Integer(2) + Y + x + Integer(1)/x, names=('y',)); (y,) =
↳ L._first_ngens(1)
>>> O = L.maximal_order()
>>> I = O.ideal(y)
>>> [f.place() for f,_ in I.factor()]
[Place (x, x*y), Place (x + 1, x*y)]

>>> # needs sage.rings.finite_rings sage.rings.function_field
>>> K = FunctionField(GF(Integer(3)**Integer(2)), names=('x',)); (x,) = K._
↳ first_ngens(1); R = PolynomialRing(K, names=('t',)); (t,) = R._first_
↳ ngens(1)
>>> F = K.extension(t**Integer(3) + t**Integer(2) - x**Integer(4), names=('y',
↳ )); (y,) = F._first_ngens(1)
>>> Oinf = F.maximal_order_infinite()
>>> I = Oinf.ideal(Integer(1)/x)
>>> I.factor()
(Ideal (1/x^3*y^2) of Maximal infinite order of Function field
in y defined by y^3 + y^2 + 2*x^4)^3
>>> J = I.factor()[Integer(0)][Integer(0)]
>>> J.is_prime()
True
>>> J.place()
Place (1/x, 1/x^3*y^2)

>>> # needs sage.rings.function_field
>>> K = FunctionField(GF(Integer(2)), names=('x',)); (x,) = K._first_ngens(1);
↳ _ = K['Y']; (Y,) = _._first_ngens(1)
>>> L = K.extension(Y**Integer(2) + Y + x + Integer(1)/x, names=('y',)); (y,) =
↳ L._first_ngens(1)
>>> Oinf = L.maximal_order_infinite()
>>> I = Oinf.ideal(Integer(1)/x)
>>> I.factor()
(Ideal (1/x*y) of Maximal infinite order of Function field in y
defined by y^2 + y + (x^2 + 1)/x)^2
>>> J = I.factor()[Integer(0)][Integer(0)]
>>> J.is_prime()
True

```

(continues on next page)

(continued from previous page)

```
>>> J.place()
Place (1/x, 1/x*y)
```

ring()

Return the ring to which this ideal belongs.

EXAMPLES:

```
sage: K.<x> = FunctionField(GF(7))
sage: O = K.equation_order()
sage: I = O.ideal(x, x^2, x^2 + x)
sage: I.ring()
Maximal order of Rational function field in x over Finite Field of size 7
```

```
>>> from sage.all import *
>>> K = FunctionField(GF(Integer(7)), names=('x',)); (x,) = K._first_ngens(1)
>>> O = K.equation_order()
>>> I = O.ideal(x, x**Integer(2), x**Integer(2) + x)
>>> I.ring()
Maximal order of Rational function field in x over Finite Field of size 7
```

class sage.rings.function_field.ideal.**FunctionFieldIdealInfinite**(ring)

Bases: *FunctionFieldIdeal*

Base class of ideals of maximal infinite orders

class sage.rings.function_field.ideal.**FunctionFieldIdealInfinite_module**(ring, module)

Bases: *FunctionFieldIdealInfinite*, *Ideal_generic*

A fractional ideal specified by a finitely generated module over the integers of the base field.

INPUT:

- ring – order in a function field
- module – module

EXAMPLES:

```
sage: # needs sage.rings.function_field
sage: K.<x> = FunctionField(QQ); R.<y> = K[]
sage: L.<y> = K.extension(y^2 - x^3 - 1)
sage: O = L.equation_order()
sage: O.ideal(y)
Ideal (x^3 + 1, -y) of Order in Function field in y defined by y^2 - x^3 - 1
```

```
>>> from sage.all import *
>>> # needs sage.rings.function_field
>>> K = FunctionField(QQ, names=('x',)); (x,) = K._first_ngens(1); R = K['y']; (y,
↪) = R._first_ngens(1)
>>> L = K.extension(y**Integer(2) - x**Integer(3) - Integer(1), names=('y',)); (y,
↪) = L._first_ngens(1)
>>> O = L.equation_order()
>>> O.ideal(y)
Ideal (x^3 + 1, -y) of Order in Function field in y defined by y^2 - x^3 - 1
```

module()

Return the module over the maximal order of the base field that underlies this ideal.

The formation of the module is compatible with the vector space corresponding to the function field.

EXAMPLES:

```
sage: K.<x> = FunctionField(GF(7))
sage: O = K.maximal_order(); O
Maximal order of Rational function field in x over Finite Field of size 7
sage: K.polynomial_ring()
Univariate Polynomial Ring in x over
Rational function field in x over Finite Field of size 7
sage: I = O.ideal([x^2 + 1, x*(x^2+1)])
sage: I.gens()
(x^2 + 1,)
sage: I.module() #_
↳needs sage.modules
Free module of degree 1 and rank 1 over
Maximal order of Rational function field in x over Finite Field of size 7
Echelon basis matrix:
[x^2 + 1]
sage: V, from_V, to_V = K.vector_space(); V #_
↳needs sage.modules
Vector space of dimension 1 over
Rational function field in x over Finite Field of size 7
sage: I.module().is_submodule(V) #_
↳needs sage.modules
True
```

```
>>> from sage.all import *
>>> K = FunctionField(GF(Integer(7)), names=('x',)); (x,) = K._first_ngens(1)
>>> O = K.maximal_order(); O
Maximal order of Rational function field in x over Finite Field of size 7
>>> K.polynomial_ring()
Univariate Polynomial Ring in x over
Rational function field in x over Finite Field of size 7
>>> I = O.ideal([x**Integer(2) + Integer(1), x*(x**Integer(2)+Integer(1))])
>>> I.gens()
(x^2 + 1,)
>>> I.module() #_
↳needs sage.modules
Free module of degree 1 and rank 1 over
Maximal order of Rational function field in x over Finite Field of size 7
Echelon basis matrix:
[x^2 + 1]
>>> V, from_V, to_V = K.vector_space(); V #_
↳needs sage.modules
Vector space of dimension 1 over
Rational function field in x over Finite Field of size 7
>>> I.module().is_submodule(V) #_
↳needs sage.modules
True
```

class sage.rings.function_field.ideal.**FunctionFieldIdeal_module**(ring, module)

Bases: *FunctionFieldIdeal*, *Ideal_generic*

A fractional ideal specified by a finitely generated module over the integers of the base field.

INPUT:

- ring – an order in a function field
- module – a module of the order

EXAMPLES:

An ideal in an extension of a rational function field:

```
sage: # needs sage.rings.function_field
sage: K.<x> = FunctionField(QQ); R.<y> = K[]
sage: L.<y> = K.extension(y^2 - x^3 - 1)
sage: O = L.equation_order()
sage: I = O.ideal(y)
sage: I
Ideal (x^3 + 1, -y) of Order in Function field in y defined by y^2 - x^3 - 1
sage: I^2
Ideal (x^3 + 1, (-x^3 - 1)*y) of Order in Function field in y defined by y^2 - x^
↪ 3 - 1
```

```
>>> from sage.all import *
>>> # needs sage.rings.function_field
>>> K = FunctionField(QQ, names=('x',)); (x,) = K._first_ngens(1); R = K['y']; (y,
↪ ) = R._first_ngens(1)
>>> L = K.extension(y**Integer(2) - x**Integer(3) - Integer(1), names=('y',)); (y,
↪ ) = L._first_ngens(1)
>>> O = L.equation_order()
>>> I = O.ideal(y)
>>> I
Ideal (x^3 + 1, -y) of Order in Function field in y defined by y^2 - x^3 - 1
>>> I**Integer(2)
Ideal (x^3 + 1, (-x^3 - 1)*y) of Order in Function field in y defined by y^2 - x^
↪ 3 - 1
```

gen(*i*)

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

EXAMPLES:

```
sage: # needs sage.rings.function_field
sage: K.<x> = FunctionField(QQ); R.<y> = K[]
sage: L.<y> = K.extension(y^2 - x^3 - 1)
sage: O = L.equation_order()
sage: I = O.ideal(x^2 + 1)
sage: I.gen(1)
(x^2 + 1)*y
```

```
>>> from sage.all import *
>>> # needs sage.rings.function_field
>>> K = FunctionField(QQ, names=('x',)); (x,) = K._first_ngens(1); R = K['y'];
```

(continues on next page)

(continued from previous page)

```

↪ (y,) = R._first_ngens(1)
>>> L = K.extension(y**Integer(2) - x**Integer(3) - Integer(1), names=('y',));
↪ (y,) = L._first_ngens(1)
>>> O = L.equation_order()
>>> I = O.ideal(x**Integer(2) + Integer(1))
>>> I.gen(Integer(1))
(x^2 + 1)*y

```

gens()

Return a set of generators of this ideal.

EXAMPLES:

```

sage: # needs sage.rings.function_field
sage: K.<x> = FunctionField(QQ); R.<y> = K[]
sage: L.<y> = K.extension(y^2 - x^3 - 1)
sage: O = L.equation_order()
sage: I = O.ideal(x^2 + 1)
sage: I.gens()
(x^2 + 1, (x^2 + 1)*y)

```

```

>>> from sage.all import *
>>> # needs sage.rings.function_field
>>> K = FunctionField(QQ, names=('x',)); (x,) = K._first_ngens(1); R = K['y'];
↪ (y,) = R._first_ngens(1)
>>> L = K.extension(y**Integer(2) - x**Integer(3) - Integer(1), names=('y',));
↪ (y,) = L._first_ngens(1)
>>> O = L.equation_order()
>>> I = O.ideal(x**Integer(2) + Integer(1))
>>> I.gens()
(x^2 + 1, (x^2 + 1)*y)

```

intersection(*other*)Return the intersection of this ideal and *other*.

EXAMPLES:

```

sage: # needs sage.rings.function_field
sage: K.<x> = FunctionField(QQ); R.<y> = K[]
sage: L.<y> = K.extension(y^2 - x^3 - 1)
sage: O = L.equation_order()
sage: I = O.ideal(y^3); J = O.ideal(y^2)
sage: Z = I.intersection(J); Z
Ideal (x^6 + 2*x^3 + 1, (-x^3 - 1)*y) of Order in Function field in y defined_
↪ by y^2 - x^3 - 1
sage: y^2 in Z
False
sage: y^3 in Z
True

```

```

>>> from sage.all import *
>>> # needs sage.rings.function_field

```

(continues on next page)

(continued from previous page)

```

>>> K = FunctionField(QQ, names=('x',)); (x,) = K._first_ngens(1); R = K['y'];
↳ (y,) = R._first_ngens(1)
>>> L = K.extension(y**Integer(2) - x**Integer(3) - Integer(1), names=('y',));
↳ (y,) = L._first_ngens(1)
>>> O = L.equation_order()
>>> I = O.ideal(y**Integer(3)); J = O.ideal(y**Integer(2))
>>> Z = I.intersection(J); Z
Ideal (x^6 + 2*x^3 + 1, (-x^3 - 1)*y) of Order in Function field in y defined
↳ by y^2 - x^3 - 1
>>> y**Integer(2) in Z
False
>>> y**Integer(3) in Z
True

```

module()

Return the module over the maximal order of the base field that underlies this ideal.

The formation of the module is compatible with the vector space corresponding to the function field.

OUTPUT: a module over the maximal order of the base field of the ideal

EXAMPLES:

```

sage: # needs sage.rings.function_field
sage: K.<x> = FunctionField(QQ); R.<y> = K[]
sage: L.<y> = K.extension(y^2 - x^3 - 1)
sage: O = L.equation_order(); O
Order in Function field in y defined by y^2 - x^3 - 1
sage: I = O.ideal(x^2 + 1)
sage: I.gens()
(x^2 + 1, (x^2 + 1)*y)
sage: I.module()
Free module of degree 2 and rank 2 over Maximal order of Rational function
↳ field in x over Rational Field
Echelon basis matrix:
[x^2 + 1      0]
[      0 x^2 + 1]
sage: V, from_V, to_V = L.vector_space(); V
Vector space of dimension 2 over Rational function field in x over Rational
↳ Field
sage: I.module().is_submodule(V)
True

```

```

>>> from sage.all import *
>>> # needs sage.rings.function_field
>>> K = FunctionField(QQ, names=('x',)); (x,) = K._first_ngens(1); R = K['y'];
↳ (y,) = R._first_ngens(1)
>>> L = K.extension(y**Integer(2) - x**Integer(3) - Integer(1), names=('y',));
↳ (y,) = L._first_ngens(1)
>>> O = L.equation_order(); O
Order in Function field in y defined by y^2 - x^3 - 1
>>> I = O.ideal(x**Integer(2) + Integer(1))
>>> I.gens()

```

(continues on next page)

(continued from previous page)

```

(x^2 + 1, (x^2 + 1)*y)
>>> I.module()
Free module of degree 2 and rank 2 over Maximal order of Rational function_
↪field in x over Rational Field
Echelon basis matrix:
[x^2 + 1      0]
[      0 x^2 + 1]
>>> V, from_V, to_V = L.vector_space(); V
Vector space of dimension 2 over Rational function field in x over Rational_
↪Field
>>> I.module().is_submodule(V)
True

```

ngens()

Return the number of generators in the basis.

EXAMPLES:

```

sage: # needs sage.rings.function_field
sage: K.<x> = FunctionField(QQ); R.<y> = K[]
sage: L.<y> = K.extension(y^2 - x^3 - 1)
sage: O = L.equation_order()
sage: I = O.ideal(x^2 + 1)
sage: I.ngens()
2

```

```

>>> from sage.all import *
>>> # needs sage.rings.function_field
>>> K = FunctionField(QQ, names=('x',)); (x,) = K._first_ngens(1); R = K['y'];
↪ (y,) = R._first_ngens(1)
>>> L = K.extension(y**Integer(2) - x**Integer(3) - Integer(1), names=('y',));
↪ (y,) = L._first_ngens(1)
>>> O = L.equation_order()
>>> I = O.ideal(x**Integer(2) + Integer(1))
>>> I.ngens()
2

```

class sage.rings.function_field.ideal.IdealMonoid(R)

Bases: UniqueRepresentation, Parent

The monoid of ideals in orders of function fields.

INPUT:

- R – order

EXAMPLES:

```

sage: K.<x> = FunctionField(GF(2))
sage: O = K.maximal_order()
sage: M = O.ideal_monoid(); M
Monoid of ideals of Maximal order of Rational function field in x over Finite_
↪Field of size 2

```

```
>>> from sage.all import *
>>> K = FunctionField(GF(Integer(2)), names=('x',)); (x,) = K._first_ngens(1)
>>> O = K.maximal_order()
>>> M = O.ideal_monoid(); M
Monoid of ideals of Maximal order of Rational function field in x over Finite_
↪Field of size 2
```

ring()

Return the ring of which this is the ideal monoid.

EXAMPLES:

```
sage: K.<x> = FunctionField(GF(2))
sage: O = K.maximal_order()
sage: M = O.ideal_monoid(); M.ring() is O
True
```

```
>>> from sage.all import *
>>> K = FunctionField(GF(Integer(2)), names=('x',)); (x,) = K._first_ngens(1)
>>> O = K.maximal_order()
>>> M = O.ideal_monoid(); M.ring() is O
True
```


IDEALS OF FUNCTION FIELDS: RATIONAL

```
class sage.rings.function_field.ideal_rational.FunctionFieldIdealInfinite_rational(ring,
                                                                                    gen)
```

Bases: *FunctionFieldIdealInfinite*

Fractional ideal of the maximal order of rational function field.

INPUT:

- ring – infinite maximal order
- gen – generator

Note that the infinite maximal order is a principal ideal domain.

EXAMPLES:

```
sage: K.<x> = FunctionField(GF(2))
sage: Oinf = K.maximal_order_infinite()
sage: Oinf.ideal(x)
Ideal (x) of Maximal infinite order of Rational function field in x over Finite_
↪Field of size 2
```

```
>>> from sage.all import *
>>> K = FunctionField(GF(Integer(2)), names=('x',)); (x,) = K._first_ngens(1)
>>> Oinf = K.maximal_order_infinite()
>>> Oinf.ideal(x)
Ideal (x) of Maximal infinite order of Rational function field in x over Finite_
↪Field of size 2
```

gen()

Return the generator of this principal ideal.

EXAMPLES:

```
sage: # needs sage.rings.finite_rings
sage: K.<x> = FunctionField(GF(2))
sage: Oinf = K.maximal_order_infinite()
sage: I = Oinf.ideal((x+1)/(x^3+x), (x^2+1)/x^4)
sage: I.gen()
1/x^2
```

```
>>> from sage.all import *
>>> # needs sage.rings.finite_rings
```

(continues on next page)

(continued from previous page)

```

>>> K = FunctionField(GF(Integer(2)), names=('x',)); (x,) = K._first_ngens(1)
>>> Oinf = K.maximal_order_infinite()
>>> I = Oinf.ideal((x+Integer(1))/(x**Integer(3)+x),
↳ (x**Integer(2)+Integer(1))/x**Integer(4))
>>> I.gen()
1/x^2

```

gens()

Return the generator of this principal ideal.

EXAMPLES:

```

sage: # needs sage.rings.finite_rings
sage: K.<x> = FunctionField(GF(2))
sage: Oinf = K.maximal_order_infinite()
sage: I = Oinf.ideal((x+1)/(x^3+x), (x^2+1)/x^4)
sage: I.gens()
(1/x^2,)

```

```

>>> from sage.all import *
>>> # needs sage.rings.finite_rings
>>> K = FunctionField(GF(Integer(2)), names=('x',)); (x,) = K._first_ngens(1)
>>> Oinf = K.maximal_order_infinite()
>>> I = Oinf.ideal((x+Integer(1))/(x**Integer(3)+x),
↳ (x**Integer(2)+Integer(1))/x**Integer(4))
>>> I.gens()
(1/x^2,)

```

gens_over_base()

Return the generator of this ideal as a rank one module over the infinite maximal order.

EXAMPLES:

```

sage: # needs sage.rings.finite_rings
sage: K.<x> = FunctionField(GF(2))
sage: Oinf = K.maximal_order_infinite()
sage: I = Oinf.ideal((x+1)/(x^3+x), (x^2+1)/x^4)
sage: I.gens_over_base()
(1/x^2,)

```

```

>>> from sage.all import *
>>> # needs sage.rings.finite_rings
>>> K = FunctionField(GF(Integer(2)), names=('x',)); (x,) = K._first_ngens(1)
>>> Oinf = K.maximal_order_infinite()
>>> I = Oinf.ideal((x+Integer(1))/(x**Integer(3)+x),
↳ (x**Integer(2)+Integer(1))/x**Integer(4))
>>> I.gens_over_base()
(1/x^2,)

```

is_prime()

Return True if this ideal is a prime ideal.

EXAMPLES:

```
sage: # needs sage.rings.finite_rings
sage: K.<x> = FunctionField(GF(2))
sage: Oinf = K.maximal_order_infinite()
sage: I = Oinf.ideal(x/(x^2 + 1))
sage: I.is_prime()
True
```

```
>>> from sage.all import *
>>> # needs sage.rings.finite_rings
>>> K = FunctionField(GF(Integer(2)), names=('x',)); (x,) = K._first_ngens(1)
>>> Oinf = K.maximal_order_infinite()
>>> I = Oinf.ideal(x/(x**Integer(2) + Integer(1)))
>>> I.is_prime()
True
```

valuation (*ideal*)

Return the valuation of *ideal* at this prime ideal.

INPUT:

- *ideal* – fractional ideal

EXAMPLES:

```
sage: F.<x> = FunctionField(QQ)
sage: O = F.maximal_order_infinite()
sage: p = O.ideal(1/x)
sage: p.valuation(O.ideal(x/(x+1)))
0
sage: p.valuation(O.ideal(0))
+Infinity
```

```
>>> from sage.all import *
>>> F = FunctionField(QQ, names=('x',)); (x,) = F._first_ngens(1)
>>> O = F.maximal_order_infinite()
>>> p = O.ideal(Integer(1)/x)
>>> p.valuation(O.ideal(x/(x+Integer(1))))
0
>>> p.valuation(O.ideal(Integer(0)))
+Infinity
```

class sage.rings.function_field.ideal_rational.**FunctionFieldIdeal_rational** (*ring*, *gen*)

Bases: *FunctionFieldIdeal*

Fractional ideals of the maximal order of a rational function field.

INPUT:

- *ring* – the maximal order of the rational function field
- *gen* – generator of the ideal, an element of the function field

EXAMPLES:

```
sage: K.<x> = FunctionField(QQ)
sage: O = K.maximal_order()
```

(continues on next page)

(continued from previous page)

```
sage: I = O.ideal(1/(x^2+x)); I
Ideal (1/(x^2 + x)) of Maximal order of Rational function field in x over
↪Rational Field
```

```
>>> from sage.all import *
>>> K = FunctionField(QQ, names=('x',)); (x,) = K._first_ngens(1)
>>> O = K.maximal_order()
>>> I = O.ideal(Integer(1)/(x**Integer(2)+x)); I
Ideal (1/(x^2 + x)) of Maximal order of Rational function field in x over
↪Rational Field
```

denominator()

Return the denominator of this fractional ideal.

EXAMPLES:

```
sage: F.<x> = FunctionField(QQ)
sage: O = F.maximal_order()
sage: I = O.ideal(x/(x^2+1))
sage: I.denominator()
x^2 + 1
```

```
>>> from sage.all import *
>>> F = FunctionField(QQ, names=('x',)); (x,) = F._first_ngens(1)
>>> O = F.maximal_order()
>>> I = O.ideal(x/(x**Integer(2)+Integer(1)))
>>> I.denominator()
x^2 + 1
```

gen()

Return the unique generator of this ideal.

EXAMPLES:

```
sage: # needs sage.rings.finite_rings
sage: K.<x> = FunctionField(GF(4))
sage: O = K.maximal_order()
sage: I = O.ideal(x^2 + x)
sage: I.gen()
x^2 + x
```

```
>>> from sage.all import *
>>> # needs sage.rings.finite_rings
>>> K = FunctionField(GF(Integer(4)), names=('x',)); (x,) = K._first_ngens(1)
>>> O = K.maximal_order()
>>> I = O.ideal(x**Integer(2) + x)
>>> I.gen()
x^2 + x
```

gens()

Return the tuple of the unique generator of this ideal.

EXAMPLES:

```

sage: # needs sage.rings.finite_rings
sage: K.<x> = FunctionField(GF(4))
sage: O = K.maximal_order()
sage: I = O.ideal(x^2 + x)
sage: I.gens()
(x^2 + x,)

```

```

>>> from sage.all import *
>>> # needs sage.rings.finite_rings
>>> K = FunctionField(GF(Integer(4)), names=('x',)); (x,) = K._first_ngens(1)
>>> O = K.maximal_order()
>>> I = O.ideal(x**Integer(2) + x)
>>> I.gens()
(x^2 + x,)

```

`gens_over_base()`

Return the generator of this ideal as a rank one module over the maximal order.

EXAMPLES:

```

sage: # needs sage.rings.finite_rings
sage: K.<x> = FunctionField(GF(4))
sage: O = K.maximal_order()
sage: I = O.ideal(x^2 + x)
sage: I.gens_over_base()
(x^2 + x,)

```

```

>>> from sage.all import *
>>> # needs sage.rings.finite_rings
>>> K = FunctionField(GF(Integer(4)), names=('x',)); (x,) = K._first_ngens(1)
>>> O = K.maximal_order()
>>> I = O.ideal(x**Integer(2) + x)
>>> I.gens_over_base()
(x^2 + x,)

```

`is_prime()`

Return True if this is a prime ideal.

EXAMPLES:

```

sage: K.<x> = FunctionField(QQ)
sage: O = K.maximal_order()
sage: I = O.ideal(x^3 + x^2)
sage: [f.is_prime() for f,m in I.factor()]
↪needs sage.libs.pari
[True, True]

```

```

>>> from sage.all import *
>>> K = FunctionField(QQ, names=('x',)); (x,) = K._first_ngens(1)
>>> O = K.maximal_order()
>>> I = O.ideal(x**Integer(3) + x**Integer(2))
>>> [f.is_prime() for f,m in I.factor()]

```

(continues on next page)

(continued from previous page)

```
↪needs sage.libs.pari
[True, True]
```

module()

Return the module, that is the ideal viewed as a module over the ring.

EXAMPLES:

```
sage: K.<x> = FunctionField(QQ)
sage: O = K.maximal_order()
sage: I = O.ideal(x^3 + x^2)
sage: I.module()
↪                                     # needs sage.modules
Free module of degree 1 and rank 1 over Maximal order of Rational
function field in x over Rational Field
Echelon basis matrix:
[x^3 + x^2]
sage: J = 0*I
sage: J.module()
↪                                     # needs sage.modules
Free module of degree 1 and rank 0 over Maximal order of Rational
function field in x over Rational Field
Echelon basis matrix:
[]
```

```
>>> from sage.all import *
>>> K = FunctionField(QQ, names=('x',)); (x,) = K._first_ngens(1)
>>> O = K.maximal_order()
>>> I = O.ideal(x**Integer(3) + x**Integer(2))
>>> I.module()
↪                                     # needs sage.modules
Free module of degree 1 and rank 1 over Maximal order of Rational
function field in x over Rational Field
Echelon basis matrix:
[x^3 + x^2]
>>> J = Integer(0)*I
>>> J.module()
↪                                     # needs sage.modules
Free module of degree 1 and rank 0 over Maximal order of Rational
function field in x over Rational Field
Echelon basis matrix:
[]
```

valuation(ideal)

Return the valuation of the ideal at this prime ideal.

INPUT:

- ideal – fractional ideal

EXAMPLES:

```
sage: F.<x> = FunctionField(QQ)
sage: O = F.maximal_order()
```

(continues on next page)

(continued from previous page)

```
sage: I = O.ideal(x^2*(x^2+x+1)^3)
sage: [f.valuation(I) for f, _ in I.factor()] #_
↪needs sage.libs.pari
[2, 3]
```

```
>>> from sage.all import *
>>> F = FunctionField(QQ, names=('x',)); (x,) = F._first_ngens(1)
>>> O = F.maximal_order()
>>> I = O.ideal(x**Integer(2)*(x**Integer(2)+x+Integer(1))**Integer(3))
>>> [f.valuation(I) for f, _ in I.factor()] #_
↪needs sage.libs.pari
[2, 3]
```


IDEALS OF FUNCTION FIELDS: EXTENSION

```
class sage.rings.function_field.ideal_polymod.FunctionFieldIdealInfinite_polymod(ring,  
                                                                                   ideal)
```

Bases: *FunctionFieldIdealInfinite*

Ideals of the infinite maximal order of an algebraic function field.

INPUT:

- *ring* – infinite maximal order of the function field
- *ideal* – ideal in the inverted function field

EXAMPLES:

```
sage: # needs sage.rings.finite_rings
sage: K.<x> = FunctionField(GF(3^2)); R.<t> = PolynomialRing(K)
sage: F.<y> = K.extension(t^3 + t^2 - x^4)
sage: Oinf = F.maximal_order_infinite()
sage: Oinf.ideal(1/y)
Ideal (1/x^4*y^2) of Maximal infinite order of Function field
in y defined by y^3 + y^2 + 2*x^4
```

```
>>> from sage.all import *
>>> # needs sage.rings.finite_rings
>>> K = FunctionField(GF(Integer(3)**Integer(2)), names=('x',)); (x,) = K._first_
↳ ngens(1); R = PolynomialRing(K, names=('t',)); (t,) = R._first_ngens(1)
>>> F = K.extension(t**Integer(3) + t**Integer(2) - x**Integer(4), names=('y',));
↳ (y,) = F._first_ngens(1)
>>> Oinf = F.maximal_order_infinite()
>>> Oinf.ideal(Integer(1)/y)
Ideal (1/x^4*y^2) of Maximal infinite order of Function field
in y defined by y^3 + y^2 + 2*x^4
```

gens()

Return a set of generators of this ideal.

EXAMPLES:

```
sage: # needs sage.rings.finite_rings
sage: K.<x> = FunctionField(GF(3^2)); R.<t> = PolynomialRing(K)
sage: F.<y> = K.extension(t^3 + t^2 - x^4)
sage: Oinf = F.maximal_order_infinite()
sage: I = Oinf.ideal(x + y)
```

(continues on next page)

(continued from previous page)

```
sage: I.gens()
(x, y, 1/x^2*y^2)
```

```
sage: # needs sage.rings.finite_rings
sage: K.<x> = FunctionField(GF(2)); _.<Y> = K[]
sage: L.<y> = K.extension(Y^2 + Y + x + 1/x)
sage: Oinf = L.maximal_order_infinite()
sage: I = Oinf.ideal(x + y)
sage: I.gens()
(x, y)
```

```
>>> from sage.all import *
>>> # needs sage.rings.finite_rings
>>> K = FunctionField(GF(Integer(3)**Integer(2)), names=('x',)); (x,) = K._
↳ first_ngens(1); R = PolynomialRing(K, names=('t',)); (t,) = R._first_
↳ ngens(1)
>>> F = K.extension(t**Integer(3) + t**Integer(2) - x**Integer(4), names=('y',
↳ )); (y,) = F._first_ngens(1)
>>> Oinf = F.maximal_order_infinite()
>>> I = Oinf.ideal(x + y)
>>> I.gens()
(x, y, 1/x^2*y^2)

>>> # needs sage.rings.finite_rings
>>> K = FunctionField(GF(Integer(2)), names=('x',)); (x,) = K._first_ngens(1);
↳ _ = K['Y']; (Y,) = _.first_ngens(1)
>>> L = K.extension(Y**Integer(2) + Y + x + Integer(1)/x, names=('y',)); (y,) =
↳ L._first_ngens(1)
>>> Oinf = L.maximal_order_infinite()
>>> I = Oinf.ideal(x + y)
>>> I.gens()
(x, y)
```

gens_over_base()

Return a set of generators of this ideal.

EXAMPLES:

```
sage: # needs sage.rings.finite_rings
sage: K.<x> = FunctionField(GF(3^2)); _.<t> = K[]
sage: F.<y> = K.extension(t^3 + t^2 - x^4)
sage: Oinf = F.maximal_order_infinite()
sage: I = Oinf.ideal(x + y)
sage: I.gens_over_base()
(x, y, 1/x^2*y^2)
```

```
>>> from sage.all import *
>>> # needs sage.rings.finite_rings
>>> K = FunctionField(GF(Integer(3)**Integer(2)), names=('x',)); (x,) = K._
↳ first_ngens(1); _ = K['t']; (t,) = _.first_ngens(1)
>>> F = K.extension(t**Integer(3) + t**Integer(2) - x**Integer(4), names=('y',
↳ )); (y,) = F._first_ngens(1)
```

(continues on next page)

(continued from previous page)

```

>>> Oinf = F.maximal_order_infinite()
>>> I = Oinf.ideal(x + y)
>>> I.gens_over_base()
(x, y, 1/x^2*y^2)

```

gens_two()

Return a set of at most two generators of this ideal.

EXAMPLES:

```

sage: # needs sage.rings.finite_rings
sage: K.<x> = FunctionField(GF(3^2)); R.<t> = PolynomialRing(K)
sage: F.<y> = K.extension(t^3 + t^2 - x^4)
sage: Oinf = F.maximal_order_infinite()
sage: I = Oinf.ideal(x + y)
sage: I.gens_two()
(x, y)

```

```

sage: # needs sage.rings.finite_rings
sage: K.<x> = FunctionField(GF(2)); L.<Y> = K[]
sage: L.<y> = K.extension(Y^2 + Y + x + 1/x)
sage: Oinf = L.maximal_order_infinite()
sage: I = Oinf.ideal(x + y)
sage: I.gens_two()
(x, )

```

```

>>> from sage.all import *
>>> # needs sage.rings.finite_rings
>>> K = FunctionField(GF(Integer(3)**Integer(2)), names=('x',)); (x,) = K._
↳ first_ngens(1); R = PolynomialRing(K, names=('t',)); (t,) = R._first_
↳ ngens(1)
>>> F = K.extension(t**Integer(3) + t**Integer(2) - x**Integer(4), names=('y',
↳ )); (y,) = F._first_ngens(1)
>>> Oinf = F.maximal_order_infinite()
>>> I = Oinf.ideal(x + y)
>>> I.gens_two()
(x, y)

>>> # needs sage.rings.finite_rings
>>> K = FunctionField(GF(Integer(2)), names=('x',)); (x,) = K._first_ngens(1);
↳ _ = K['Y']; (Y,) = _._first_ngens(1)
>>> L = K.extension(Y**Integer(2) + Y + x + Integer(1)/x, names=('y',)); (y,) =
↳ L._first_ngens(1)
>>> Oinf = L.maximal_order_infinite()
>>> I = Oinf.ideal(x + y)
>>> I.gens_two()
(x, )

```

ideal_below()

Return a set of generators of this ideal.

EXAMPLES:

```

sage: # needs sage.rings.finite_rings
sage: K.<x> = FunctionField(GF(3^2)); _.<t> = K[]
sage: F.<y> = K.extension(t^3 + t^2 - x^4)
sage: Oinf = F.maximal_order_infinite()
sage: I = Oinf.ideal(1/y^2)
sage: I.ideal_below()
Ideal (x^3) of Maximal order of Rational function field
in x over Finite Field in z2 of size 3^2

```

```

>>> from sage.all import *
>>> # needs sage.rings.finite_rings
>>> K = FunctionField(GF(Integer(3)**Integer(2)), names=('x',)); (x,) = K._
↳first_ngens(1); _ = K['t']; (t,) = _.first_ngens(1)
>>> F = K.extension(t**Integer(3) + t**Integer(2) - x**Integer(4), names=('y',
↳)); (y,) = F._first_ngens(1)
>>> Oinf = F.maximal_order_infinite()
>>> I = Oinf.ideal(Integer(1)/y**Integer(2))
>>> I.ideal_below()
Ideal (x^3) of Maximal order of Rational function field
in x over Finite Field in z2 of size 3^2

```

is_prime()

Return True if this ideal is a prime ideal.

EXAMPLES:

```

sage: # needs sage.rings.finite_rings
sage: K.<x> = FunctionField(GF(3^2)); _.<t> = PolynomialRing(K)
sage: F.<y> = K.extension(t^3 + t^2 - x^4)
sage: Oinf = F.maximal_order_infinite()
sage: I = Oinf.ideal(1/x)
sage: I.factor()
(Ideal (1/x^3*y^2) of Maximal infinite order of Function field
in y defined by y^3 + y^2 + 2*x^4)^3
sage: I.is_prime()
False
sage: J = I.factor()[0][0]
sage: J.is_prime()
True

sage: # needs sage.rings.finite_rings
sage: K.<x> = FunctionField(GF(2)); _.<Y> = K[]
sage: L.<y> = K.extension(Y^2 + Y + x + 1/x)
sage: Oinf = L.maximal_order_infinite()
sage: I = Oinf.ideal(1/x)
sage: I.factor()
(Ideal (1/x*y) of Maximal infinite order of Function field in y
defined by y^2 + y + (x^2 + 1)/x)^2
sage: I.is_prime()
False
sage: J = I.factor()[0][0]
sage: J.is_prime()
True

```

```

>>> from sage.all import *
>>> # needs sage.rings.finite_rings
>>> K = FunctionField(GF(Integer(3)**Integer(2)), names=('x',)); (x,) = K._
↳ first_ngens(1); _ = PolynomialRing(K, names=('t',)); (t,) = _._first_
↳ ngens(1)
>>> F = K.extension(t**Integer(3) + t**Integer(2) - x**Integer(4), names=('y',
↳ )); (y,) = F._first_ngens(1)
>>> Oinf = F.maximal_order_infinite()
>>> I = Oinf.ideal(Integer(1)/x)
>>> I.factor()
(Ideal (1/x^3*y^2) of Maximal infinite order of Function field
in y defined by y^3 + y^2 + 2*x^4)^3
>>> I.is_prime()
False
>>> J = I.factor()[Integer(0)][Integer(0)]
>>> J.is_prime()
True

>>> # needs sage.rings.finite_rings
>>> K = FunctionField(GF(Integer(2)), names=('x',)); (x,) = K._first_ngens(1);
↳ _ = K['Y']; (Y,) = _._first_ngens(1)
>>> L = K.extension(Y**Integer(2) + Y + x + Integer(1)/x, names=('y',)); (y,) =
↳ L._first_ngens(1)
>>> Oinf = L.maximal_order_infinite()
>>> I = Oinf.ideal(Integer(1)/x)
>>> I.factor()
(Ideal (1/x*y) of Maximal infinite order of Function field in y
defined by y^2 + y + (x^2 + 1)/x)^2
>>> I.is_prime()
False
>>> J = I.factor()[Integer(0)][Integer(0)]
>>> J.is_prime()
True

```

prime_below()

Return the prime of the base order that underlies this prime ideal.

EXAMPLES:

```

sage: # needs sage.rings.finite_rings
sage: K.<x> = FunctionField(GF(3^2)); _.<t> = PolynomialRing(K)
sage: F.<y> = K.extension(t^3 + t^2 - x^4)
sage: Oinf = F.maximal_order_infinite()
sage: I = Oinf.ideal(1/x)
sage: I.factor()
(Ideal (1/x^3*y^2) of Maximal infinite order of Function field
in y defined by y^3 + y^2 + 2*x^4)^3
sage: J = I.factor()[0][0]
sage: J.is_prime()
True
sage: J.prime_below()
Ideal (1/x) of Maximal infinite order of Rational function field
in x over Finite Field in z2 of size 3^2

```

(continues on next page)

(continued from previous page)

```

sage: # needs sage.rings.finite_rings
sage: K.<x> = FunctionField(GF(2)); _.<Y> = K[]
sage: L.<y> = K.extension(Y^2 + Y + x + 1/x)
sage: Oinf = L.maximal_order_infinite()
sage: I = Oinf.ideal(1/x)
sage: I.factor()
(Ideal (1/x*y) of Maximal infinite order of Function field in y
defined by y^2 + y + (x^2 + 1)/x)^2
sage: J = I.factor()[0][0]
sage: J.is_prime()
True
sage: J.prime_below()
Ideal (1/x) of Maximal infinite order of Rational function field in x
over Finite Field of size 2

```

```

>>> from sage.all import *
>>> # needs sage.rings.finite_rings
>>> K = FunctionField(GF(Integer(3)**Integer(2)), names=('x',)); (x,) = K._
↪ first_ngens(1); _ = PolynomialRing(K, names=('t',)); (t,) = _._first_
↪ ngens(1)
>>> F = K.extension(t**Integer(3) + t**Integer(2) - x**Integer(4), names=('y',
↪ )); (y,) = F._first_ngens(1)
>>> Oinf = F.maximal_order_infinite()
>>> I = Oinf.ideal(Integer(1)/x)
>>> I.factor()
(Ideal (1/x^3*y^2) of Maximal infinite order of Function field
in y defined by y^3 + y^2 + 2*x^4)^3
>>> J = I.factor()[Integer(0)][Integer(0)]
>>> J.is_prime()
True
>>> J.prime_below()
Ideal (1/x) of Maximal infinite order of Rational function field
in x over Finite Field in z2 of size 3^2

>>> # needs sage.rings.finite_rings
>>> K = FunctionField(GF(Integer(2)), names=('x',)); (x,) = K._first_ngens(1);
↪ _ = K['Y']; (Y,) = _._first_ngens(1)
>>> L = K.extension(Y**Integer(2) + Y + x + Integer(1)/x, names=('y',)); (y,) ↪
↪ = L._first_ngens(1)
>>> Oinf = L.maximal_order_infinite()
>>> I = Oinf.ideal(Integer(1)/x)
>>> I.factor()
(Ideal (1/x*y) of Maximal infinite order of Function field in y
defined by y^2 + y + (x^2 + 1)/x)^2
>>> J = I.factor()[Integer(0)][Integer(0)]
>>> J.is_prime()
True
>>> J.prime_below()
Ideal (1/x) of Maximal infinite order of Rational function field in x
over Finite Field of size 2

```

valuation (*ideal*)

Return the valuation of `ideal` with respect to this prime ideal.

INPUT:

- `ideal` – fractional ideal

EXAMPLES:

```
sage: # needs sage.rings.finite_rings
sage: K.<x> = FunctionField(GF(2)); _.<Y> = K[]
sage: L.<y> = K.extension(Y^2 + Y + x + 1/x)
sage: Oinf = L.maximal_order_infinite()
sage: I = Oinf.ideal(y)
sage: [f.valuation(I) for f,_ in I.factor()]
[-1]
```

```
>>> from sage.all import *
>>> # needs sage.rings.finite_rings
>>> K = FunctionField(GF(Integer(2)), names=('x',)); (x,) = K._first_ngens(1);
↪ _ = K['Y']; (Y,) = _._first_ngens(1)
>>> L = K.extension(Y**Integer(2) + Y + x + Integer(1)/x, names=('y',)); (y,) =
↪ L._first_ngens(1)
>>> Oinf = L.maximal_order_infinite()
>>> I = Oinf.ideal(y)
>>> [f.valuation(I) for f,_ in I.factor()]
[-1]
```

```
class sage.rings.function_field.ideal_polymod.FunctionFieldIdeal_global(ring, hnf,
                                                                    denominator=1)
```

Bases: *FunctionFieldIdeal_polymod*

Fractional ideals of canonical function fields.

INPUT:

- `ring` – order in a function field
- `hnf` – matrix in hermite normal form
- `denominator` – denominator

The rows of `hnf` is a basis of the ideal, which itself is `denominator` times the fractional ideal.

EXAMPLES:

```
sage: # needs sage.rings.finite_rings
sage: K.<x> = FunctionField(GF(2)); R.<y> = K[]
sage: L.<y> = K.extension(y^2 - x^3*y - x)
sage: O = L.maximal_order()
sage: O.ideal(y)
Ideal (y) of Maximal order of Function field in y defined by y^2 + x^3*y + x
```

```
>>> from sage.all import *
>>> # needs sage.rings.finite_rings
>>> K = FunctionField(GF(Integer(2)), names=('x',)); (x,) = K._first_ngens(1); R_
↪ = K['y']; (y,) = R._first_ngens(1)
>>> L = K.extension(y**Integer(2) - x**Integer(3)*y - x, names=('y',)); (y,) = L._
(continues on next page)
```

(continued from previous page)

```

↪first_ngens(1)
>>> O = L.maximal_order()
>>> O.ideal(y)
Ideal (y) of Maximal order of Function field in y defined by y^2 + x^3*y + x

```

gens()

Return a set of generators of this ideal.

This provides whatever set of generators as quickly as possible.

EXAMPLES:

```

sage: # needs sage.rings.finite_rings
sage: K.<x> = FunctionField(GF(2)); _.<Y> = K[]
sage: L.<y> = K.extension(Y^2 - x^3*Y - x)
sage: O = L.maximal_order()
sage: I = O.ideal(x + y)
sage: I.gens()
(x^4 + x^2 + x, y + x)

sage: # needs sage.rings.finite_rings
sage: L.<y> = K.extension(Y^2 + Y + x + 1/x)
sage: O = L.maximal_order()
sage: I = O.ideal(x + y)
sage: I.gens()
(x^3 + 1, y + x)

```

```

>>> from sage.all import *
>>> # needs sage.rings.finite_rings
>>> K = FunctionField(GF(Integer(2)), names=('x',)); (x,) = K._first_ngens(1);
↪ _ = K['Y']; (Y,) = _._first_ngens(1)
>>> L = K.extension(Y**Integer(2) - x**Integer(3)*Y - x, names=('y',)); (y,) ↪
↪ = L._first_ngens(1)
>>> O = L.maximal_order()
>>> I = O.ideal(x + y)
>>> I.gens()
(x^4 + x^2 + x, y + x)

>>> # needs sage.rings.finite_rings
>>> L = K.extension(Y**Integer(2) + Y + x + Integer(1)/x, names=('y',)); (y,) ↪
↪ = L._first_ngens(1)
>>> O = L.maximal_order()
>>> I = O.ideal(x + y)
>>> I.gens()
(x^3 + 1, y + x)

```

gens_two()

Return two generators of this fractional ideal.

If the ideal is principal, one generator *may* be returned.

EXAMPLES:


```

sage: # needs sage.rings.finite_rings
sage: K.<x> = FunctionField(GF(2)); _.<t> = K[]
sage: F.<y> = K.extension(t^3 - x^2*(x^2 + x + 1)^2)
sage: O = F.maximal_order()
sage: I = O.ideal(y)
sage: I # indirect doctest
Ideal (y) of Maximal order of Function field
in y defined by y^3 + x^6 + x^4 + x^2
sage: ~I # indirect doctest
Ideal ((1/(x^6 + x^4 + x^2))*y^2) of Maximal order of Function field
in y defined by y^3 + x^6 + x^4 + x^2

sage: # needs sage.rings.finite_rings
sage: K.<x> = FunctionField(GF(2)); _.<Y> = K[]
sage: L.<y> = K.extension(Y^2 + Y + x + 1/x)
sage: O = L.maximal_order()
sage: I = O.ideal(y)
sage: I # indirect doctest
Ideal (y) of Maximal order of Function field in y
defined by y^2 + y + (x^2 + 1)/x
sage: ~I # indirect doctest
Ideal ((x/(x^2 + 1))*y + x/(x^2 + 1)) of Maximal order
of Function field in y defined by y^2 + y + (x^2 + 1)/x

```

```

>>> from sage.all import *
>>> # needs sage.rings.finite_rings
>>> K = FunctionField(GF(Integer(2)), names=('x',)); (x,) = K._first_ngens(1);
↳ _ = K['t']; (t,) = _._first_ngens(1)
>>> F = K.extension(t**Integer(3) - x**Integer(2)*(x**Integer(2) + x +
↳ Integer(1))**Integer(2), names=('y',)); (y,) = F._first_ngens(1)
>>> O = F.maximal_order()
>>> I = O.ideal(y)
>>> I # indirect doctest
Ideal (y) of Maximal order of Function field
in y defined by y^3 + x^6 + x^4 + x^2
>>> ~I # indirect doctest
Ideal ((1/(x^6 + x^4 + x^2))*y^2) of Maximal order of Function field
in y defined by y^3 + x^6 + x^4 + x^2

>>> # needs sage.rings.finite_rings
>>> K = FunctionField(GF(Integer(2)), names=('x',)); (x,) = K._first_ngens(1);
↳ _ = K['Y']; (Y,) = _._first_ngens(1)
>>> L = K.extension(Y**Integer(2) + Y + x + Integer(1)/x, names=('y',)); (y,) =
↳ L._first_ngens(1)
>>> O = L.maximal_order()
>>> I = O.ideal(y)
>>> I # indirect doctest
Ideal (y) of Maximal order of Function field in y
defined by y^2 + y + (x^2 + 1)/x
>>> ~I # indirect doctest
Ideal ((x/(x^2 + 1))*y + x/(x^2 + 1)) of Maximal order
of Function field in y defined by y^2 + y + (x^2 + 1)/x

```

```
class sage.rings.function_field.ideal_polymod.FunctionFieldIdeal_polymod(ring, hnf,
                                                                           denominator=1)
```

Bases: *FunctionFieldIdeal*

Fractional ideals of algebraic function fields.

INPUT:

- ring – order in a function field
- hnf – matrix in hermite normal form
- denominator – denominator

The rows of hnf is a basis of the ideal, which itself is denominator times the fractional ideal.

EXAMPLES:

```
sage: # needs sage.rings.finite_rings
sage: K.<x> = FunctionField(GF(2)); R.<y> = K[]
sage: L.<y> = K.extension(y^2 - x^3*y - x)
sage: O = L.maximal_order()
sage: O.ideal(y)
Ideal (y) of Maximal order of Function field in y defined by y^2 + x^3*y + x
```

```
>>> from sage.all import *
>>> # needs sage.rings.finite_rings
>>> K = FunctionField(GF(Integer(2)), names=('x',)); (x,) = K._first_ngens(1); R._
↳ K['y']; (y,) = R._first_ngens(1)
>>> L = K.extension(y**Integer(2) - x**Integer(3)*y - x, names=('y',)); (y,) = L._
↳ first_ngens(1)
>>> O = L.maximal_order()
>>> O.ideal(y)
Ideal (y) of Maximal order of Function field in y defined by y^2 + x^3*y + x
```

basis_matrix()

Return the matrix of basis vectors of this ideal as a module.

The basis matrix is by definition the hermite norm form of the ideal divided by the denominator.

EXAMPLES:

```
sage: # needs sage.rings.finite_rings
sage: K.<x> = FunctionField(GF(2)); R.<t> = PolynomialRing(K)
sage: F.<y> = K.extension(t^3 - x^2*(x^2+x+1)^2)
sage: O = F.maximal_order()
sage: I = O.ideal(x, 1/y)
sage: I.denominator() * I.basis_matrix() == I.hnf()
True
```

```
>>> from sage.all import *
>>> # needs sage.rings.finite_rings
>>> K = FunctionField(GF(Integer(2)), names=('x',)); (x,) = K._first_ngens(1);
↳ R = PolynomialRing(K, names=('t',)); (t,) = R._first_ngens(1)
>>> F = K.extension(t**Integer(3) -
↳ x**Integer(2)*(x**Integer(2)+x+Integer(1))**Integer(2), names=('y',)); (y,)
↳ F._first_ngens(1)
```

(continues on next page)

(continued from previous page)

```

>>> O = F.maximal_order()
>>> I = O.ideal(x, Integer(1)/y)
>>> I.denominator() * I.basis_matrix() == I.hnf()
True

```

denominator()

Return the denominator of this fractional ideal.

EXAMPLES:

```

sage: # needs sage.rings.finite_rings
sage: K.<x> = FunctionField(GF(7)); R.<y> = K[]
sage: L.<y> = K.extension(y^2 - x^3 - 1)
sage: O = L.maximal_order()
sage: I = O.ideal(y/(y+1))
sage: d = I.denominator(); d
x^3
sage: d in O
True

```

```

>>> from sage.all import *
>>> # needs sage.rings.finite_rings
>>> K = FunctionField(GF(Integer(7)), names=('x',)); (x,) = K._first_ngens(1);
↳ R = K['y']; (y,) = R._first_ngens(1)
>>> L = K.extension(y**Integer(2) - x**Integer(3) - Integer(1), names=('y',));
↳ (y,) = L._first_ngens(1)
>>> O = L.maximal_order()
>>> I = O.ideal(y/(y+Integer(1)))
>>> d = I.denominator(); d
x^3
>>> d in O
True

```

```

sage: K.<x> = FunctionField(QQ); R.<y> = K[]
sage: L.<y> = K.extension(y^2 - x^3 - 1)
sage: O = L.maximal_order()
sage: I = O.ideal(y/(y+1))
sage: d = I.denominator(); d
x^3
sage: d in O
True

```

```

>>> from sage.all import *
>>> K = FunctionField(QQ, names=('x',)); (x,) = K._first_ngens(1); R = K['y'];
↳ (y,) = R._first_ngens(1)
>>> L = K.extension(y**Integer(2) - x**Integer(3) - Integer(1), names=('y',));
↳ (y,) = L._first_ngens(1)
>>> O = L.maximal_order()
>>> I = O.ideal(y/(y+Integer(1)))
>>> d = I.denominator(); d
x^3

```

(continues on next page)

(continued from previous page)

```
>>> d in O
True
```

gens()

Return a set of generators of this ideal.

This provides whatever set of generators as quickly as possible.

EXAMPLES:

```
sage: # needs sage.rings.finite_rings
sage: K.<x> = FunctionField(GF(2)); _.<Y> = K[]
sage: L.<y> = K.extension(Y^2 - x^3*Y - x)
sage: O = L.maximal_order()
sage: I = O.ideal(x + y)
sage: I.gens()
(x^4 + x^2 + x, y + x)

sage: # needs sage.rings.finite_rings
sage: L.<y> = K.extension(Y^2 + Y + x + 1/x)
sage: O = L.maximal_order()
sage: I = O.ideal(x + y)
sage: I.gens()
(x^3 + 1, y + x)
```

```
>>> from sage.all import *
>>> # needs sage.rings.finite_rings
>>> K = FunctionField(GF(Integer(2)), names=('x',)); (x,) = K._first_ngens(1);
↳ _ = K['Y']; (Y,) = _._first_ngens(1)
>>> L = K.extension(Y**Integer(2) - x**Integer(3)*Y - x, names=('y',)); (y,) ↳
↳ L._first_ngens(1)
>>> O = L.maximal_order()
>>> I = O.ideal(x + y)
>>> I.gens()
(x^4 + x^2 + x, y + x)

>>> # needs sage.rings.finite_rings
>>> L = K.extension(Y**Integer(2) + Y + x + Integer(1)/x, names=('y',)); (y,) ↳
↳ L._first_ngens(1)
>>> O = L.maximal_order()
>>> I = O.ideal(x + y)
>>> I.gens()
(x^3 + 1, y + x)
```

gens_over_base()

Return the generators of this ideal as a module over the maximal order of the base rational function field.

EXAMPLES:

```
sage: # needs sage.rings.finite_rings
sage: K.<x> = FunctionField(GF(2)); _.<Y> = K[]
sage: L.<y> = K.extension(Y^2 - x^3*Y - x)
sage: O = L.maximal_order()
```

(continues on next page)

(continued from previous page)

```

sage: I = O.ideal(x + y)
sage: I.gens_over_base()
(x^4 + x^2 + x, y + x)

sage: # needs sage.rings.finite_rings
sage: L.<y> = K.extension(Y^2 + Y + x + 1/x)
sage: O = L.maximal_order()
sage: I = O.ideal(x + y)
sage: I.gens_over_base()
(x^3 + 1, y + x)

```

```

>>> from sage.all import *
>>> # needs sage.rings.finite_rings
>>> K = FunctionField(GF(Integer(2)), names=('x',)); (x,) = K._first_ngens(1);
↳ _ = K['Y']; (Y,) = _._first_ngens(1)
>>> L = K.extension(Y**Integer(2) - x**Integer(3)*Y - x, names=('y',)); (y,) =
↳ L._first_ngens(1)
>>> O = L.maximal_order()
>>> I = O.ideal(x + y)
>>> I.gens_over_base()
(x^4 + x^2 + x, y + x)

>>> # needs sage.rings.finite_rings
>>> L = K.extension(Y**Integer(2) + Y + x + Integer(1)/x, names=('y',)); (y,) =
↳ L._first_ngens(1)
>>> O = L.maximal_order()
>>> I = O.ideal(x + y)
>>> I.gens_over_base()
(x^3 + 1, y + x)

```

hnf()

Return the matrix in hermite normal form representing this ideal.

See also `denominator()`

EXAMPLES:

```

sage: # needs sage.rings.finite_rings
sage: K.<x> = FunctionField(GF(7)); R.<y> = K[]
sage: L.<y> = K.extension(y^2 - x^3 - 1)
sage: O = L.maximal_order()
sage: I = O.ideal(y*(y+1)); I.hnf()
[x^6 + x^3      0]
[ x^3 + 1      1]

```

```

>>> from sage.all import *
>>> # needs sage.rings.finite_rings
>>> K = FunctionField(GF(Integer(7)), names=('x',)); (x,) = K._first_ngens(1);
↳ R = K['y']; (y,) = R._first_ngens(1)
>>> L = K.extension(y**Integer(2) - x**Integer(3) - Integer(1), names=('y',));
↳ (y,) = L._first_ngens(1)
>>> O = L.maximal_order()

```

(continues on next page)

(continued from previous page)

```
>>> I = O.ideal(y*(y+Integer(1))); I.hnf()
[x^6 + x^3      0]
[ x^3 + 1      1]
```

```
sage: K.<x> = FunctionField(QQ); R.<y> = K[]
sage: L.<y> = K.extension(y^2 - x^3 - 1)
sage: O = L.maximal_order()
sage: I = O.ideal(y*(y+1)); I.hnf()
[x^6 + x^3      0]
[ x^3 + 1      1]
```

```
>>> from sage.all import *
>>> K = FunctionField(QQ, names=('x',)); (x,) = K._first_ngens(1); R = K['y'];
↪ (y,) = R._first_ngens(1)
>>> L = K.extension(y**Integer(2) - x**Integer(3) - Integer(1), names=('y',));
↪ (y,) = L._first_ngens(1)
>>> O = L.maximal_order()
>>> I = O.ideal(y*(y+Integer(1))); I.hnf()
[x^6 + x^3      0]
[ x^3 + 1      1]
```

ideal_below()

Return the ideal below this ideal.

This is defined only for integral ideals.

EXAMPLES:

```
sage: # needs sage.rings.finite_rings
sage: K.<x> = FunctionField(GF(2)); _.<t> = K[]
sage: F.<y> = K.extension(t^3 - x^2*(x^2+x+1)^2)
sage: O = F.maximal_order()
sage: I = O.ideal(x, 1/y)
sage: I.ideal_below()
Traceback (most recent call last):
...
TypeError: not an integral ideal
sage: J = I.denominator() * I
sage: J.ideal_below()
Ideal (x^3 + x^2 + x) of Maximal order of Rational function field
in x over Finite Field of size 2

sage: # needs sage.rings.finite_rings
sage: K.<x> = FunctionField(GF(2)); _.<Y> = K[]
sage: L.<y> = K.extension(Y^2 + Y + x + 1/x)
sage: O = L.maximal_order()
sage: I = O.ideal(x, 1/y)
sage: I.ideal_below()
Traceback (most recent call last):
...
TypeError: not an integral ideal
sage: J = I.denominator() * I
```

(continues on next page)

(continued from previous page)

```

sage: J.ideal_below()
Ideal (x^3 + x) of Maximal order of Rational function field
in x over Finite Field of size 2

sage: K.<x> = FunctionField(QQ); _.<t> = K[]
sage: F.<y> = K.extension(t^3 - x^2*(x^2+x+1)^2)
sage: O = F.maximal_order()
sage: I = O.ideal(x, 1/y)
sage: I.ideal_below()
Traceback (most recent call last):
...
TypeError: not an integral ideal
sage: J = I.denominator() * I
sage: J.ideal_below()
Ideal (x^3 + x^2 + x) of Maximal order of Rational function field
in x over Rational Field

```

```

>>> from sage.all import *
>>> # needs sage.rings.finite_rings
>>> K = FunctionField(GF(Integer(2)), names=('x',)); (x,) = K._first_ngens(1);
↪ _ = K['t']; (t,) = _._first_ngens(1)
>>> F = K.extension(t**Integer(3) -
↪ x**Integer(2)*(x**Integer(2)+x+Integer(1))**Integer(2), names=('y',)); (y,) =
↪ F._first_ngens(1)
>>> O = F.maximal_order()
>>> I = O.ideal(x, Integer(1)/y)
>>> I.ideal_below()
Traceback (most recent call last):
...
TypeError: not an integral ideal
>>> J = I.denominator() * I
>>> J.ideal_below()
Ideal (x^3 + x^2 + x) of Maximal order of Rational function field
in x over Finite Field of size 2

>>> # needs sage.rings.finite_rings
>>> K = FunctionField(GF(Integer(2)), names=('x',)); (x,) = K._first_ngens(1);
↪ _ = K['Y']; (Y,) = _._first_ngens(1)
>>> L = K.extension(Y**Integer(2) + Y + x + Integer(1)/x, names=('y',)); (y,) =
↪ L._first_ngens(1)
>>> O = L.maximal_order()
>>> I = O.ideal(x, Integer(1)/y)
>>> I.ideal_below()
Traceback (most recent call last):
...
TypeError: not an integral ideal
>>> J = I.denominator() * I
>>> J.ideal_below()
Ideal (x^3 + x) of Maximal order of Rational function field
in x over Finite Field of size 2

>>> K = FunctionField(QQ, names=('x',)); (x,) = K._first_ngens(1); _ = K['t'];

```

(continues on next page)

(continued from previous page)

```

↪ (t,) = _._first_ngens(1)
>>> F = K.extension(t**Integer(3) -
↪ x**Integer(2)*(x**Integer(2)+x+Integer(1))**Integer(2), names=('y',)); (y,) ↪
↪ = F._first_ngens(1)
>>> O = F.maximal_order()
>>> I = O.ideal(x, Integer(1)/y)
>>> I.ideal_below()
Traceback (most recent call last):
...
TypeError: not an integral ideal
>>> J = I.denominator() * I
>>> J.ideal_below()
Ideal (x^3 + x^2 + x) of Maximal order of Rational function field
in x over Rational Field

```

intersect (other)

Intersect this ideal with the other ideal as fractional ideals.

INPUT:

- other – ideal

EXAMPLES:

```

sage: # needs sage.rings.finite_rings
sage: K.<x> = FunctionField(GF(2)); _.<Y> = K[]
sage: L.<y> = K.extension(Y^2 - x^3*Y - x)
sage: O = L.maximal_order()
sage: I = O.ideal(x + y)
sage: J = O.ideal(x)
sage: I.intersect(J) == I * J * (I + J)^-1
True

```

```

>>> from sage.all import *
>>> # needs sage.rings.finite_rings
>>> K = FunctionField(GF(Integer(2)), names=('x',)); (x,) = K._first_ngens(1);
↪ _ = K['Y']; (Y,) = _._first_ngens(1)
>>> L = K.extension(Y**Integer(2) - x**Integer(3)*Y - x, names=('y',)); (y,) ↪
↪ = L._first_ngens(1)
>>> O = L.maximal_order()
>>> I = O.ideal(x + y)
>>> J = O.ideal(x)
>>> I.intersect(J) == I * J * (I + J)**-Integer(1)
True

```

is_integral()

Return True if this is an integral ideal.

EXAMPLES:

```

sage: # needs sage.rings.finite_rings
sage: K.<x> = FunctionField(GF(2)); _.<t> = PolynomialRing(K)
sage: F.<y> = K.extension(t^3 - x^2*(x^2+x+1)^2)
sage: O = F.maximal_order()

```

(continues on next page)

(continued from previous page)

```

sage: I = O.ideal(x, 1/y)
sage: I.is_integral()
False
sage: J = I.denominator() * I
sage: J.is_integral()
True

sage: # needs sage.rings.finite_rings
sage: K.<x> = FunctionField(GF(2)); _.<Y> = K[]
sage: L.<y> = K.extension(Y^2 + Y + x + 1/x)
sage: O = L.maximal_order()
sage: I = O.ideal(x, 1/y)
sage: I.is_integral()
False
sage: J = I.denominator() * I
sage: J.is_integral()
True

sage: K.<x> = FunctionField(QQ); _.<t> = PolynomialRing(K)
sage: F.<y> = K.extension(t^3 - x^2*(x^2+x+1)^2)
sage: O = F.maximal_order()
sage: I = O.ideal(x, 1/y)
sage: I.is_integral()
False
sage: J = I.denominator() * I
sage: J.is_integral()
True

```

```

>>> from sage.all import *
>>> # needs sage.rings.finite_rings
>>> K = FunctionField(GF(Integer(2)), names=('x',)); (x,) = K._first_ngens(1);
↪ _ = PolynomialRing(K, names=('t',)); (t,) = _._first_ngens(1)
>>> F = K.extension(t**Integer(3) -
↪ x**Integer(2)*(x**Integer(2)+x+Integer(1))**Integer(2), names=('y',)); (y,) =
↪ F._first_ngens(1)
>>> O = F.maximal_order()
>>> I = O.ideal(x, Integer(1)/y)
>>> I.is_integral()
False
>>> J = I.denominator() * I
>>> J.is_integral()
True

>>> # needs sage.rings.finite_rings
>>> K = FunctionField(GF(Integer(2)), names=('x',)); (x,) = K._first_ngens(1);
↪ _ = K['Y']; (Y,) = _._first_ngens(1)
>>> L = K.extension(Y**Integer(2) + Y + x + Integer(1)/x, names=('y',)); (y,) =
↪ L._first_ngens(1)
>>> O = L.maximal_order()
>>> I = O.ideal(x, Integer(1)/y)
>>> I.is_integral()
False

```

(continues on next page)

(continued from previous page)

```

>>> J = I.denominator() * I
>>> J.is_integral()
True

>>> K = FunctionField(QQ, names=('x',)); (x,) = K._first_ngens(1); _ =
↳ PolynomialRing(K, names=('t',)); (t,) = _._first_ngens(1)
>>> F = K.extension(t**Integer(3) -
↳ x**Integer(2)*(x**Integer(2)+x+Integer(1))**Integer(2), names=('y',)); (y,) =
↳ F._first_ngens(1)
>>> O = F.maximal_order()
>>> I = O.ideal(x, Integer(1)/y)
>>> I.is_integral()
False
>>> J = I.denominator() * I
>>> J.is_integral()
True

```

is_prime()

Return True if this ideal is a prime ideal.

EXAMPLES:

```

sage: # needs sage.rings.finite_rings
sage: K.<x> = FunctionField(GF(2)); _.<t> = PolynomialRing(K)
sage: F.<y> = K.extension(t^3 - x^2*(x^2+x+1)^2)
sage: O = F.maximal_order()
sage: I = O.ideal(y)
sage: [f.is_prime() for f,_ in I.factor()]
[True, True]

```

```

sage: # needs sage.rings.finite_rings
sage: K.<x> = FunctionField(GF(2)); _.<Y> = K[]
sage: L.<y> = K.extension(Y^2 + Y + x + 1/x)
sage: O = L.maximal_order()
sage: I = O.ideal(y)
sage: [f.is_prime() for f,_ in I.factor()]
[True, True]

```

```

sage: K.<x> = FunctionField(QQ); _.<t> = PolynomialRing(K)
sage: F.<y> = K.extension(t^3 - x^2*(x^2+x+1)^2)
sage: O = F.maximal_order()
sage: I = O.ideal(y)
sage: [f.is_prime() for f,_ in I.factor()]
[True, True]

```

```

>>> from sage.all import *
>>> # needs sage.rings.finite_rings
>>> K = FunctionField(GF(Integer(2)), names=('x',)); (x,) = K._first_ngens(1);
↳ _ = PolynomialRing(K, names=('t',)); (t,) = _._first_ngens(1)
>>> F = K.extension(t**Integer(3) -
↳ x**Integer(2)*(x**Integer(2)+x+Integer(1))**Integer(2), names=('y',)); (y,) =
↳ F._first_ngens(1)

```

(continues on next page)

(continued from previous page)

```

>>> O = F.maximal_order()
>>> I = O.ideal(y)
>>> [f.is_prime() for f,_ in I.factor()]
[True, True]

>>> # needs sage.rings.finite_rings
>>> K = FunctionField(GF(Integer(2)), names=('x',)); (x,) = K._first_ngens(1);
↪ _ = K['Y']; (Y,) = _._first_ngens(1)
>>> L = K.extension(Y**Integer(2) + Y + x + Integer(1)/x, names=('y',)); (y,) =
↪ L._first_ngens(1)
>>> O = L.maximal_order()
>>> I = O.ideal(y)
>>> [f.is_prime() for f,_ in I.factor()]
[True, True]

>>> K = FunctionField(QQ, names=('x',)); (x,) = K._first_ngens(1); _ =
↪ PolynomialRing(K, names=('t',)); (t,) = _._first_ngens(1)
>>> F = K.extension(t**Integer(3) -
↪ x**Integer(2)*(x**Integer(2)+x+Integer(1))**Integer(2), names=('y',)); (y,) =
↪ F._first_ngens(1)
>>> O = F.maximal_order()
>>> I = O.ideal(y)
>>> [f.is_prime() for f,_ in I.factor()]
[True, True]

```

module()

Return the module, that is the ideal viewed as a module over the base maximal order.

EXAMPLES:

```

sage: # needs sage.rings.finite_rings
sage: K.<x> = FunctionField(GF(7)); R.<y> = K[]
sage: F.<y> = K.extension(y^2 - x^3 - 1)
sage: O = F.maximal_order()
sage: I = O.ideal(x, 1/y)
sage: I.module()
Free module of degree 2 and rank 2 over Maximal order
of Rational function field in x over Finite Field of size 7
Echelon basis matrix:
[      1      0]
[      0 1/(x^3 + 1)]

```

```

>>> from sage.all import *
>>> # needs sage.rings.finite_rings
>>> K = FunctionField(GF(Integer(7)), names=('x',)); (x,) = K._first_ngens(1);
↪ R = K['y']; (y,) = R._first_ngens(1)
>>> F = K.extension(y**Integer(2) - x**Integer(3) - Integer(1), names=('y',));
↪ (y,) = F._first_ngens(1)
>>> O = F.maximal_order()
>>> I = O.ideal(x, Integer(1)/y)
>>> I.module()
Free module of degree 2 and rank 2 over Maximal order

```

(continues on next page)

(continued from previous page)

```

of Rational function field in x over Finite Field of size 7
Echelon basis matrix:
[      1      0]
[      0 1/(x^3 + 1)]

```

norm()

Return the norm of this fractional ideal.

EXAMPLES:

```

sage: # needs sage.rings.finite_rings
sage: K.<x> = FunctionField(GF(2)); _.<t> = PolynomialRing(K)
sage: F.<y> = K.extension(t^3 - x^2*(x^2+x+1)^2)
sage: O = F.maximal_order()
sage: i1 = O.ideal(x)
sage: i2 = O.ideal(y)
sage: i3 = i1 * i2
sage: i3.norm() == i1.norm() * i2.norm()
True
sage: i1.norm()
x^3
sage: i1.norm() == x ** F.degree()
True
sage: i2.norm()
x^6 + x^4 + x^2
sage: i2.norm() == y.norm()
True

sage: # needs sage.rings.finite_rings
sage: K.<x> = FunctionField(GF(2)); _.<Y> = K[]
sage: L.<y> = K.extension(Y^2 + Y + x + 1/x)
sage: O = L.maximal_order()
sage: i1 = O.ideal(x)
sage: i2 = O.ideal(y)
sage: i3 = i1 * i2
sage: i3.norm() == i1.norm() * i2.norm()
True
sage: i1.norm()
x^2
sage: i1.norm() == x ** L.degree()
True
sage: i2.norm()
(x^2 + 1)/x
sage: i2.norm() == y.norm()
True

```

```

>>> from sage.all import *
>>> # needs sage.rings.finite_rings
>>> K = FunctionField(GF(Integer(2)), names=('x',)); (x,) = K._first_ngens(1);
↳ _ = PolynomialRing(K, names=('t',)); (t,) = _._first_ngens(1)
>>> F = K.extension(t**Integer(3) -
↳ x**Integer(2)*(x**Integer(2)+x+Integer(1))**Integer(2), names=('y',)); (y,) =

```

(continues on next page)

(continued from previous page)

```

↪= F._first_ngens(1)
>>> O = F.maximal_order()
>>> i1 = O.ideal(x)
>>> i2 = O.ideal(y)
>>> i3 = i1 * i2
>>> i3.norm() == i1.norm() * i2.norm()
True
>>> i1.norm()
x^3
>>> i1.norm() == x ** F.degree()
True
>>> i2.norm()
x^6 + x^4 + x^2
>>> i2.norm() == y.norm()
True

>>> # needs sage.rings.finite_rings
>>> K = FunctionField(GF(Integer(2)), names=('x',)); (x,) = K._first_ngens(1);
↪ _ = K['Y']; (Y,) = _._first_ngens(1)
>>> L = K.extension(Y**Integer(2) + Y + x + Integer(1)/x, names=('y',)); (y,) =
↪ L._first_ngens(1)
>>> O = L.maximal_order()
>>> i1 = O.ideal(x)
>>> i2 = O.ideal(y)
>>> i3 = i1 * i2
>>> i3.norm() == i1.norm() * i2.norm()
True
>>> i1.norm()
x^2
>>> i1.norm() == x ** L.degree()
True
>>> i2.norm()
(x^2 + 1)/x
>>> i2.norm() == y.norm()
True

```

prime_below()

Return the prime lying below this prime ideal.

EXAMPLES:

```

sage: # needs sage.rings.finite_rings
sage: K.<x> = FunctionField(GF(2)); _.<Y> = K[]
sage: F.<y> = K.extension(Y^3 - x^2*(x^2 + x + 1)^2)
sage: O = F.maximal_order()
sage: I = O.ideal(y)
sage: [f.prime_below() for f,_ in I.factor()]
[Ideal (x) of Maximal order of Rational function field in x
over Finite Field of size 2, Ideal (x^2 + x + 1) of Maximal order
of Rational function field in x over Finite Field of size 2]

sage: # needs sage.rings.finite_rings

```

(continues on next page)

(continued from previous page)

```

sage: K.<x> = FunctionField(GF(2)); _.<Y> = K[]
sage: L.<y> = K.extension(Y^2 + Y + x + 1/x)
sage: O = L.maximal_order()
sage: I = O.ideal(y)
sage: [f.prime_below() for f,_ in I.factor()]
[Ideal (x) of Maximal order of Rational function field in x over Finite Field
↳ of size 2,
 Ideal (x + 1) of Maximal order of Rational function field in x over Finite
↳ Field of size 2]

sage: K.<x> = FunctionField(QQ); _.<Y> = K[]
sage: F.<y> = K.extension(Y^3 - x^2*(x^2 + x + 1)^2)
sage: O = F.maximal_order()
sage: I = O.ideal(y)
sage: [f.prime_below() for f,_ in I.factor()]
[Ideal (x) of Maximal order of Rational function field in x over Rational
↳ Field,
 Ideal (x^2 + x + 1) of Maximal order of Rational function field in x over
↳ Rational Field]

```

```

>>> from sage.all import *
>>> # needs sage.rings.finite_rings
>>> K = FunctionField(GF(Integer(2)), names=('x',)); (x,) = K._first_ngens(1);
↳ _ = K['Y']; (Y,) = _._first_ngens(1)
>>> F = K.extension(Y**Integer(3) - x**Integer(2)*(x**Integer(2) + x +
↳ Integer(1))**Integer(2), names=('y',)); (y,) = F._first_ngens(1)
>>> O = F.maximal_order()
>>> I = O.ideal(y)
>>> [f.prime_below() for f,_ in I.factor()]
[Ideal (x) of Maximal order of Rational function field in x
over Finite Field of size 2, Ideal (x^2 + x + 1) of Maximal order
of Rational function field in x over Finite Field of size 2]

>>> # needs sage.rings.finite_rings
>>> K = FunctionField(GF(Integer(2)), names=('x',)); (x,) = K._first_ngens(1);
↳ _ = K['Y']; (Y,) = _._first_ngens(1)
>>> L = K.extension(Y**Integer(2) + Y + x + Integer(1)/x, names=('y',)); (y,)
↳ = L._first_ngens(1)
>>> O = L.maximal_order()
>>> I = O.ideal(y)
>>> [f.prime_below() for f,_ in I.factor()]
[Ideal (x) of Maximal order of Rational function field in x over Finite Field
↳ of size 2,
 Ideal (x + 1) of Maximal order of Rational function field in x over Finite
↳ Field of size 2]

>>> K = FunctionField(QQ, names=('x',)); (x,) = K._first_ngens(1); _ = K['Y'];
↳ (Y,) = _._first_ngens(1)
>>> F = K.extension(Y**Integer(3) - x**Integer(2)*(x**Integer(2) + x +
↳ Integer(1))**Integer(2), names=('y',)); (y,) = F._first_ngens(1)
>>> O = F.maximal_order()
>>> I = O.ideal(y)

```

(continues on next page)

(continued from previous page)

```
>>> [f.prime_below() for f,_ in I.factor()]
[Ideal (x) of Maximal order of Rational function field in x over Rational
↪Field,
 Ideal (x^2 + x + 1) of Maximal order of Rational function field in x over
↪Rational Field]
```

valuation (*ideal*)

Return the valuation of *ideal* at this prime ideal.

INPUT:

- *ideal* – fractional ideal

EXAMPLES:

```
sage: # needs sage.rings.finite_rings
sage: K.<x> = FunctionField(GF(2)); _.<t> = K[]
sage: F.<y> = K.extension(t^3 - x^2*(x^2 + x + 1)^2)
sage: O = F.maximal_order()
sage: I = O.ideal(x, (1/(x^3 + x^2 + x))*y^2)
sage: I.is_prime()
True
sage: J = O.ideal(y)
sage: I.valuation(J)
2
```

```
sage: # needs sage.rings.finite_rings
sage: K.<x> = FunctionField(GF(2)); _.<Y> = K[]
sage: L.<y> = K.extension(Y^2 + Y + x + 1/x)
sage: O = L.maximal_order()
sage: I = O.ideal(y)
sage: [f.valuation(I) for f,_ in I.factor()]
[-1, 2]
```

```
>>> from sage.all import *
>>> # needs sage.rings.finite_rings
>>> K = FunctionField(GF(Integer(2)), names=('x',)); (x,) = K._first_ngens(1);
↪ _ = K['t']; (t,) = _._first_ngens(1)
>>> F = K.extension(t**Integer(3) - x**Integer(2)*(x**Integer(2) + x +
↪ Integer(1))**Integer(2), names=('y',)); (y,) = F._first_ngens(1)
>>> O = F.maximal_order()
>>> I = O.ideal(x, (Integer(1)/(x**Integer(3) + x**Integer(2) +
↪ x))*y**Integer(2))
>>> I.is_prime()
True
>>> J = O.ideal(y)
>>> I.valuation(J)
2

>>> # needs sage.rings.finite_rings
>>> K = FunctionField(GF(Integer(2)), names=('x',)); (x,) = K._first_ngens(1);
↪ _ = K['Y']; (Y,) = _._first_ngens(1)
>>> L = K.extension(Y**Integer(2) + Y + x + Integer(1)/x, names=('y',)); (y,) =
```

(continues on next page)

(continued from previous page)

```
↪= L._first_ngens(1)
>>> O = L.maximal_order()
>>> I = O.ideal(y)
>>> [f.valuation(I) for f, _ in I.factor()]
[-1, 2]
```

The method closely follows Algorithm 4.8.17 of [Coh1993].

PLACES OF FUNCTION FIELDS

The places of a function field correspond, one-to-one, to valuation rings of the function field, each of which defines a discrete valuation for the elements of the function field. “Finite” places are in one-to-one correspondence with the prime ideals of the finite maximal order while places “at infinity” are in one-to-one correspondence with the prime ideals of the infinite maximal order.

EXAMPLES:

All rational places of a function field can be computed:

```
sage: K.<x> = FunctionField(GF(2)); _.<Y> = K[]
sage: L.<y> = K.extension(Y^3 + x + x^3*Y) #_
↪needs sage.rings.function_field
sage: L.places() #_
↪needs sage.rings.function_field
[Place (1/x, 1/x^3*y^2 + 1/x),
 Place (1/x, 1/x^3*y^2 + 1/x^2*y + 1),
 Place (x, y)]
```

```
>>> from sage.all import *
>>> K = FunctionField(GF(Integer(2)), names=('x',)); (x,) = K._first_ngens(1); _ = K[
↪'Y']; (Y,) = _._first_ngens(1)
>>> L = K.extension(Y**Integer(3) + x + x**Integer(3)*Y, names=('y',)); (y,) = L._
↪first_ngens(1) # needs sage.rings.function_field
>>> L.places() #_
↪needs sage.rings.function_field
[Place (1/x, 1/x^3*y^2 + 1/x),
 Place (1/x, 1/x^3*y^2 + 1/x^2*y + 1),
 Place (x, y)]
```

The residue field associated with a place is given as an extension of the constant field:

```
sage: F.<x> = FunctionField(GF(2))
sage: O = F.maximal_order()
sage: p = O.ideal(x^2 + x + 1).place() #_
↪needs sage.libs.pari
sage: k, fr_k, to_k = p.residue_field() #_
↪needs sage.libs.pari sage.rings.function_field
sage: k #_
↪needs sage.libs.pari sage.rings.function_field
Finite Field in z2 of size 2^2
```

```

>>> from sage.all import *
>>> F = FunctionField(GF(Integer(2)), names=('x',)); (x,) = F._first_ngens(1)
>>> O = F.maximal_order()
>>> p = O.ideal(x*Integer(2) + x + Integer(1)).place()
↪ # needs sage.libs.pari
>>> k, fr_k, to_k = p.residue_field()
↪ needs sage.libs.pari sage.rings.function_field
>>> k
↪ needs sage.libs.pari sage.rings.function_field
Finite Field in z2 of size 2^2

```

The homomorphisms are between the valuation ring and the residue field:

```

sage: fr_k
↪ needs sage.libs.pari sage.rings.function_field
Ring morphism:
  From: Finite Field in z2 of size 2^2
  To:   Valuation ring at Place (x^2 + x + 1)
sage: to_k
↪ needs sage.libs.pari sage.rings.function_field
Ring morphism:
  From: Valuation ring at Place (x^2 + x + 1)
  To:   Finite Field in z2 of size 2^2

```

```

>>> from sage.all import *
>>> fr_k
↪ needs sage.libs.pari sage.rings.function_field
Ring morphism:
  From: Finite Field in z2 of size 2^2
  To:   Valuation ring at Place (x^2 + x + 1)
>>> to_k
↪ needs sage.libs.pari sage.rings.function_field
Ring morphism:
  From: Valuation ring at Place (x^2 + x + 1)
  To:   Finite Field in z2 of size 2^2

```

AUTHORS:

- Kwanky Lee (2017-04-30): initial version
- Brent Baccala (2019-12-20): function fields of characteristic zero

```
class sage.rings.function_field.place.FunctionFieldPlace(parent, prime)
```

Bases: `Element`

Places of function fields.

INPUT:

- `parent` – place set of a function field
- `prime` – prime ideal associated with the place

EXAMPLES:

```

sage: K.<x> = FunctionField(GF(2)); _.<Y> = K[]
sage: L.<y> = K.extension(Y^3 + x + x^3*Y) #_
↪needs sage.rings.function_field
sage: L.places_finite()[0] #_
↪needs sage.rings.function_field
Place (x, y)

```

```

>>> from sage.all import *
>>> K = FunctionField(GF(Integer(2)), names=('x',)); (x,) = K._first_ngens(1); _
↪= K['Y']; (Y,) = _._first_ngens(1)
>>> L = K.extension(Y**Integer(3) + x + x**Integer(3)*Y, names=('y',)); (y,) = L._
↪first_ngens(1) # needs sage.rings.function_field
>>> L.places_finite()[Integer(0)] #_
↪ # needs sage.rings.function_field
Place (x, y)

```

divisor (multiplicity=1)

Return the prime divisor corresponding to the place.

EXAMPLES:

```

sage: # needs sage.rings.function_field
sage: K.<x> = FunctionField(GF(5)); R.<Y> = PolynomialRing(K)
sage: F.<y> = K.extension(Y^2 - x^3 - 1)
sage: O = F.maximal_order()
sage: I = O.ideal(x + 1, y)
sage: P = I.place()
sage: P.divisor()
Place (x + 1, y)

```

```

>>> from sage.all import *
>>> # needs sage.rings.function_field
>>> K = FunctionField(GF(Integer(5)), names=('x',)); (x,) = K._first_ngens(1);
↪ R = PolynomialRing(K, names=('Y',)); (Y,) = R._first_ngens(1)
>>> F = K.extension(Y**Integer(2) - x**Integer(3) - Integer(1), names=('y',));
↪ (y,) = F._first_ngens(1)
>>> O = F.maximal_order()
>>> I = O.ideal(x + Integer(1), y)
>>> P = I.place()
>>> P.divisor()
Place (x + 1, y)

```

function_field()

Return the function field to which the place belongs.

EXAMPLES:

```

sage: K.<x> = FunctionField(GF(2)); _.<Y> = K[]
sage: L.<y> = K.extension(Y^3 + x^3*Y + x) #_
↪needs sage.rings.function_field
sage: p = L.places()[0] #_
↪needs sage.rings.function_field
sage: p.function_field() == L #_

```

(continues on next page)

(continued from previous page)

```
↪needs sage.rings.function_field
True
```

```
>>> from sage.all import *
>>> K = FunctionField(GF(Integer(2)), names=('x',)); (x,) = K._first_ngens(1);
↪ _ = K['Y']; (Y,) = _._first_ngens(1)
>>> L = K.extension(Y**Integer(3) + x**Integer(3)*Y + x, names=('y',)); (y,) ↪
↪= L._first_ngens(1) # needs sage.rings.function_field
>>> p = L.places()[Integer(0)]
↪ # needs sage.rings.function_field
>>> p.function_field() == L
↪ #
↪needs sage.rings.function_field
True
```

prime_ideal()

Return the prime ideal associated with the place.

EXAMPLES:

```
sage: K.<x> = FunctionField(GF(2)); _.<Y> = K[]
sage: L.<y> = K.extension(Y^3 + x^3*Y + x)
↪needs sage.rings.function_field
sage: p = L.places()[0]
↪needs sage.rings.function_field
sage: p.prime_ideal()
↪needs sage.rings.function_field
Ideal (1/x^3*y^2 + 1/x) of Maximal infinite order of Function field
in y defined by y^3 + x^3*y + x
```

```
>>> from sage.all import *
>>> K = FunctionField(GF(Integer(2)), names=('x',)); (x,) = K._first_ngens(1);
↪ _ = K['Y']; (Y,) = _._first_ngens(1)
>>> L = K.extension(Y**Integer(3) + x**Integer(3)*Y + x, names=('y',)); (y,) ↪
↪= L._first_ngens(1) # needs sage.rings.function_field
>>> p = L.places()[Integer(0)]
↪ # needs sage.rings.function_field
>>> p.prime_ideal()
↪needs sage.rings.function_field
Ideal (1/x^3*y^2 + 1/x) of Maximal infinite order of Function field
in y defined by y^3 + x^3*y + x
```

class sage.rings.function_field.place.PlaceSet (*field*)

Bases: UniqueRepresentation, Parent

Sets of Places of function fields.

INPUT:

- field – function field

EXAMPLES:

```
sage: K.<x> = FunctionField(GF(2)); _.<Y> = K[]
sage: L.<y> = K.extension(Y^3 + x^3*Y + x)
```

(continues on next page)

(continued from previous page)

```

↪needs sage.rings.function_field
sage: L.place_set() #_
↪needs sage.rings.function_field
Set of places of Function field in y defined by  $y^3 + x^3y + x$ 

>>> from sage.all import *
>>> K = FunctionField(GF(Integer(2)), names=('x',)); (x,) = K._first_ngens(1); _
↪= K['Y']; (Y,) = _._first_ngens(1)
>>> L = K.extension(Y**Integer(3) + x**Integer(3)*Y + x, names=('y',)); (y,) = L._
↪first_ngens(1) # needs sage.rings.function_field
>>> L.place_set() #_
↪needs sage.rings.function_field
Set of places of Function field in y defined by  $y^3 + x^3y + x$ 

```

Elementalias of *FunctionFieldPlace***function_field()**

Return the function field to which this place set belongs.

EXAMPLES:

```

sage: # needs sage.rings.function_field
sage: K.<x> = FunctionField(GF(2)); _.<Y> = K[]
sage: L.<y> = K.extension(Y^3 + x^3*Y + x)
sage: PS = L.place_set()
sage: PS.function_field() == L
True

```

```

>>> from sage.all import *
>>> # needs sage.rings.function_field
>>> K = FunctionField(GF(Integer(2)), names=('x',)); (x,) = K._first_ngens(1);
↪ _ = K['Y']; (Y,) = _._first_ngens(1)
>>> L = K.extension(Y**Integer(3) + x**Integer(3)*Y + x, names=('y',)); (y,)
↪ = L._first_ngens(1)
>>> PS = L.place_set()
>>> PS.function_field() == L
True

```


PLACES OF FUNCTION FIELDS: RATIONAL

class sage.rings.function_field.place_rational.**FunctionFieldPlace_rational** (*parent, prime*)

Bases: *FunctionFieldPlace*

Places of rational function fields.

degree ()

Return the degree of the place.

EXAMPLES:

```
sage: F.<x> = FunctionField(GF(2))
sage: O = F.maximal_order()
sage: i = O.ideal(x^2 + x + 1)
sage: p = i.place()
sage: p.degree()
2
```

```
>>> from sage.all import *
>>> F = FunctionField(GF(Integer(2)), names=('x',)); (x,) = F._first_ngens(1)
>>> O = F.maximal_order()
>>> i = O.ideal(x**Integer(2) + x + Integer(1))
>>> p = i.place()
>>> p.degree()
2
```

is_infinite_place ()

Return True if the place is at infinite.

EXAMPLES:

```
sage: F.<x> = FunctionField(GF(2))
sage: F.places()
[Place (1/x), Place (x), Place (x + 1)]
sage: [p.is_infinite_place() for p in F.places()]
[True, False, False]
```

```
>>> from sage.all import *
>>> F = FunctionField(GF(Integer(2)), names=('x',)); (x,) = F._first_ngens(1)
>>> F.places()
[Place (1/x), Place (x), Place (x + 1)]
>>> [p.is_infinite_place() for p in F.places()]
[True, False, False]
```

local_uniformizer()

Return a local uniformizer of the place.

EXAMPLES:

```
sage: F.<x> = FunctionField(GF(2))
sage: F.places()
[Place (1/x), Place (x), Place (x + 1)]
sage: [p.local_uniformizer() for p in F.places()]
[1/x, x, x + 1]
```

```
>>> from sage.all import *
>>> F = FunctionField(GF(Integer(2)), names=('x',)); (x,) = F._first_ngens(1)
>>> F.places()
[Place (1/x), Place (x), Place (x + 1)]
>>> [p.local_uniformizer() for p in F.places()]
[1/x, x, x + 1]
```

residue_field(name=None)

Return the residue field of the place.

EXAMPLES:

```
sage: F.<x> = FunctionField(GF(2))
sage: O = F.maximal_order()
sage: p = O.ideal(x^2 + x + 1).place()
sage: k, fr_k, to_k = p.residue_field() #_
↪needs sage.rings.function_field
sage: k #_
↪needs sage.rings.function_field
Finite Field in z2 of size 2^2
sage: fr_k #_
↪needs sage.rings.function_field
Ring morphism:
  From: Finite Field in z2 of size 2^2
  To:   Valuation ring at Place (x^2 + x + 1)
sage: to_k #_
↪needs sage.rings.function_field
Ring morphism:
  From: Valuation ring at Place (x^2 + x + 1)
  To:   Finite Field in z2 of size 2^2
```

```
>>> from sage.all import *
>>> F = FunctionField(GF(Integer(2)), names=('x',)); (x,) = F._first_ngens(1)
>>> O = F.maximal_order()
>>> p = O.ideal(x*Integer(2) + x + Integer(1)).place()
>>> k, fr_k, to_k = p.residue_field() #_
↪needs sage.rings.function_field
>>> k #_
↪needs sage.rings.function_field
Finite Field in z2 of size 2^2
>>> fr_k #_
↪needs sage.rings.function_field
Ring morphism:
```

(continues on next page)

(continued from previous page)

```

From: Finite Field in z2 of size 2^2
To: Valuation ring at Place (x^2 + x + 1)
>>> to_k #_
↪needs sage.rings.function_field
Ring morphism:
From: Valuation ring at Place (x^2 + x + 1)
To: Finite Field in z2 of size 2^2

```

valuation_ring()

Return the valuation ring at the place.

EXAMPLES:

```

sage: K.<x> = FunctionField(GF(2)); _.<Y> = K[]
sage: L.<y> = K.extension(Y^2 + Y + x + 1/x) #_
↪needs sage.rings.function_field
sage: p = L.places_finite()[0] #_
↪needs sage.rings.function_field
sage: p.valuation_ring() #_
↪needs sage.rings.function_field
Valuation ring at Place (x, x*y)

```

```

>>> from sage.all import *
>>> K = FunctionField(GF(Integer(2)), names=('x',)); (x,) = K._first_ngens(1);
↪ _ = K['Y']; (Y,) = _._first_ngens(1)
>>> L = K.extension(Y**Integer(2) + Y + x + Integer(1)/x, names=('y',)); (y,) #_
↪ = L._first_ngens(1) # needs sage.rings.function_field
>>> p = L.places_finite()[Integer(0)] #_
↪ # needs sage.rings.function_field
>>> p.valuation_ring() #_
↪needs sage.rings.function_field
Valuation ring at Place (x, x*y)

```


PLACES OF FUNCTION FIELDS: EXTENSION

class sage.rings.function_field.place_polymod.**FunctionFieldPlace_polymod**(parent, prime)

Bases: *FunctionFieldPlace*

Places of extensions of function fields.

degree()

Return the degree of the place.

EXAMPLES:

```
sage: # needs sage.rings.finite_rings
sage: K.<x> = FunctionField(GF(2)); _.<Y> = K[]
sage: L.<y> = K.extension(Y^3 + x^3*Y + x)
sage: OK = K.maximal_order()
sage: OL = L.maximal_order()
sage: p = OK.ideal(x^2 + x + 1)
sage: dec = OL.decomposition(p)
sage: q = dec[0][0].place()
sage: q.degree()
2
```

```
>>> from sage.all import *
>>> # needs sage.rings.finite_rings
>>> K = FunctionField(GF(Integer(2)), names=('x',)); (x,) = K._first_ngens(1);
↳ _ = K['Y']; (Y,) = _._first_ngens(1)
>>> L = K.extension(Y**Integer(3) + x**Integer(3)*Y + x, names=('y',)); (y,) =
↳ L._first_ngens(1)
>>> OK = K.maximal_order()
>>> OL = L.maximal_order()
>>> p = OK.ideal(x**Integer(2) + x + Integer(1))
>>> dec = OL.decomposition(p)
>>> q = dec[Integer(0)][Integer(0)].place()
>>> q.degree()
2
```

gaps()

Return the gap sequence for the place.

EXAMPLES:

```
sage: # needs sage.rings.finite_rings
sage: K.<x> = FunctionField(GF(4)); _.<Y> = K[]
```

(continues on next page)

(continued from previous page)

```

sage: L.<y> = K.extension(Y^3 + x^3*Y + x)
sage: O = L.maximal_order()
sage: p = O.ideal(x,y).place()
sage: p.gaps() # a Weierstrass place
[1, 2, 4]

sage: K.<x> = FunctionField(GF(2)); _.<Y> = K[] #_
↳needs sage.rings.finite_rings
sage: L.<y> = K.extension(Y^3 + x^3 * Y + x) #_
↳needs sage.rings.finite_rings
sage: [p.gaps() for p in L.places()] #_
↳needs sage.rings.finite_rings
[[1, 2, 4], [1, 2, 4], [1, 2, 4]]

```

```

>>> from sage.all import *
>>> # needs sage.rings.finite_rings
>>> K = FunctionField(GF(Integer(4)), names=('x',)); (x,) = K._first_ngens(1);
↳ _ = K['Y']; (Y,) = _._first_ngens(1)
>>> L = K.extension(Y**Integer(3) + x**Integer(3)*Y + x, names=('y',)); (y,) =
↳ L._first_ngens(1)
>>> O = L.maximal_order()
>>> p = O.ideal(x,y).place()
>>> p.gaps() # a Weierstrass place
[1, 2, 4]

>>> K = FunctionField(GF(Integer(2)), names=('x',)); (x,) = K._first_ngens(1);
↳ _ = K['Y']; (Y,) = _._first_ngens(1) # needs sage.rings.finite_rings
>>> L = K.extension(Y**Integer(3) + x**Integer(3) * Y + x, names=('y',)); (y,
↳ ) = L._first_ngens(1) # needs sage.rings.finite_rings
>>> [p.gaps() for p in L.places()] #_
↳needs sage.rings.finite_rings
[[1, 2, 4], [1, 2, 4], [1, 2, 4]]

```

is_infinite_place()

Return True if the place is above the unique infinite place of the underlying rational function field.

EXAMPLES:

```

sage: # needs sage.rings.finite_rings
sage: K.<x> = FunctionField(GF(2)); _.<Y> = K[]
sage: L.<y> = K.extension(Y^3 + x^3*Y + x)
sage: pls = L.places()
sage: [p.is_infinite_place() for p in pls]
[True, True, False]
sage: [p.place_below() for p in pls]
[Place (1/x), Place (1/x), Place (x)]

```

```

>>> from sage.all import *
>>> # needs sage.rings.finite_rings
>>> K = FunctionField(GF(Integer(2)), names=('x',)); (x,) = K._first_ngens(1);
↳ _ = K['Y']; (Y,) = _._first_ngens(1)
>>> L = K.extension(Y**Integer(3) + x**Integer(3)*Y + x, names=('y',)); (y,) =

```

(continues on next page)

(continued from previous page)

```

↪ = L._first_ngens(1)
>>> pls = L.places()
>>> [p.is_infinite_place() for p in pls]
[True, True, False]
>>> [p.place_below() for p in pls]
[Place (1/x), Place (1/x), Place (x)]

```

local_uniformizer()

Return an element of the function field that has a simple zero at the place.

EXAMPLES:

```

sage: # needs sage.rings.finite_rings
sage: K.<x> = FunctionField(GF(4)); _.<Y> = K[]
sage: L.<y> = K.extension(Y^3 + x^3*Y + x)
sage: pls = L.places()
sage: [p.local_uniformizer().valuation(p) for p in pls]
[1, 1, 1, 1, 1]

```

```

>>> from sage.all import *
>>> # needs sage.rings.finite_rings
>>> K = FunctionField(GF(Integer(4)), names=('x',)); (x,) = K._first_ngens(1);
↪ _ = K['Y']; (Y,) = _._first_ngens(1)
>>> L = K.extension(Y**Integer(3) + x**Integer(3)*Y + x, names=('y',)); (y,) =
↪ L._first_ngens(1)
>>> pls = L.places()
>>> [p.local_uniformizer().valuation(p) for p in pls]
[1, 1, 1, 1, 1]

```

place_below()

Return the place lying below the place.

EXAMPLES:

```

sage: # needs sage.rings.finite_rings
sage: K.<x> = FunctionField(GF(2)); _.<Y> = K[]
sage: L.<y> = K.extension(Y^3 + x^3*Y + x)
sage: OK = K.maximal_order()
sage: OL = L.maximal_order()
sage: p = OK.ideal(x^2 + x + 1)
sage: dec = OL.decomposition(p)
sage: q = dec[0][0].place()
sage: q.place_below()
Place (x^2 + x + 1)

```

```

>>> from sage.all import *
>>> # needs sage.rings.finite_rings
>>> K = FunctionField(GF(Integer(2)), names=('x',)); (x,) = K._first_ngens(1);
↪ _ = K['Y']; (Y,) = _._first_ngens(1)
>>> L = K.extension(Y**Integer(3) + x**Integer(3)*Y + x, names=('y',)); (y,) =
↪ L._first_ngens(1)
>>> OK = K.maximal_order()

```

(continues on next page)

(continued from previous page)

```

>>> OL = L.maximal_order()
>>> p = OK.ideal(x**Integer(2) + x + Integer(1))
>>> dec = OL.decomposition(p)
>>> q = dec[Integer(0)][Integer(0)].place()
>>> q.place_below()
Place (x^2 + x + 1)

```

relative_degree()

Return the relative degree of the place.

EXAMPLES:

```

sage: # needs sage.rings.finite_rings
sage: K.<x> = FunctionField(GF(2)); _.<Y> = K[]
sage: L.<y> = K.extension(Y^3 + x^3*Y + x)
sage: OK = K.maximal_order()
sage: OL = L.maximal_order()
sage: p = OK.ideal(x^2 + x + 1)
sage: dec = OL.decomposition(p)
sage: q = dec[0][0].place()
sage: q.relative_degree()
1

```

```

>>> from sage.all import *
>>> # needs sage.rings.finite_rings
>>> K = FunctionField(GF(Integer(2)), names=('x',)); (x,) = K._first_ngens(1);
↪ _ = K['Y']; (Y,) = _._first_ngens(1)
>>> L = K.extension(Y**Integer(3) + x**Integer(3)*Y + x, names=('y',)); (y,) =
↪ L._first_ngens(1)
>>> OK = K.maximal_order()
>>> OL = L.maximal_order()
>>> p = OK.ideal(x**Integer(2) + x + Integer(1))
>>> dec = OL.decomposition(p)
>>> q = dec[Integer(0)][Integer(0)].place()
>>> q.relative_degree()
1

```

residue_field(name=None)

Return the residue field of the place.

INPUT:

- name – string; name of the generator of the residue field

OUTPUT:

- a field isomorphic to the residue field
- a ring homomorphism from the valuation ring to the field
- a ring homomorphism from the field to the valuation ring

EXAMPLES:

```

sage: # needs sage.rings.finite_rings
sage: K.<x> = FunctionField(GF(2)); _.<Y> = K[]

```

(continues on next page)

(continued from previous page)

```

sage: L.<y> = K.extension(Y^2 + Y + x + 1/x)
sage: p = L.places_finite()[0]
sage: k, fr_k, to_k = p.residue_field()
sage: k
Finite Field of size 2
sage: fr_k
Ring morphism:
  From: Finite Field of size 2
  To:   Valuation ring at Place (x, x*y)
sage: to_k
Ring morphism:
  From: Valuation ring at Place (x, x*y)
  To:   Finite Field of size 2
sage: to_k(y)
Traceback (most recent call last):
...
TypeError: y fails to convert into the map's domain
Valuation ring at Place (x, x*y)...
sage: to_k(1/y)
0
sage: to_k(y/(1+y))
1

```

```

>>> from sage.all import *
>>> # needs sage.rings.finite_rings
>>> K = FunctionField(GF(Integer(2)), names=('x',)); (x,) = K._first_ngens(1);
↪ _ = K['Y']; (Y,) = _._first_ngens(1)
>>> L = K.extension(Y**Integer(2) + Y + x + Integer(1)/x, names=('y',)); (y,) ↪
↪ = L._first_ngens(1)
>>> p = L.places_finite()[Integer(0)]
>>> k, fr_k, to_k = p.residue_field()
>>> k
Finite Field of size 2
>>> fr_k
Ring morphism:
  From: Finite Field of size 2
  To:   Valuation ring at Place (x, x*y)
>>> to_k
Ring morphism:
  From: Valuation ring at Place (x, x*y)
  To:   Finite Field of size 2
>>> to_k(y)
Traceback (most recent call last):
...
TypeError: y fails to convert into the map's domain
Valuation ring at Place (x, x*y)...
>>> to_k(Integer(1)/y)
0
>>> to_k(y/(Integer(1)+y))
1

```

valuation_ring()

Return the valuation ring at the place.

EXAMPLES:

```
sage: # needs sage.rings.finite_rings
sage: K.<x> = FunctionField(GF(2)); _.<Y> = K[]
sage: L.<y> = K.extension(Y^2 + Y + x + 1/x)
sage: p = L.places_finite()[0]
sage: p.valuation_ring()
Valuation ring at Place (x, x*y)
```

```
>>> from sage.all import *
>>> # needs sage.rings.finite_rings
>>> K = FunctionField(GF(Integer(2)), names=('x',)); (x,) = K._first_ngens(1);
↪ _ = K['Y']; (Y,) = _._first_ngens(1)
>>> L = K.extension(Y**Integer(2) + Y + x + Integer(1)/x, names=('y',)); (y,) ↪
↪ = L._first_ngens(1)
>>> p = L.places_finite()[Integer(0)]
>>> p.valuation_ring()
Valuation ring at Place (x, x*y)
```


DIVISORS OF FUNCTION FIELDS

Sage allows extensive computations with divisors on function fields.

EXAMPLES:

The divisor of an element of the function field is the formal sum of poles and zeros of the element with multiplicities:

```
sage: K.<x> = FunctionField(GF(2)); R.<t> = K[]
sage: L.<y> = K.extension(t^3 + x^3*t + x)
sage: f = x/(y+1)
sage: f.divisor()
- Place (1/x, 1/x^3*y^2 + 1/x)
+ Place (1/x, 1/x^3*y^2 + 1/x^2*y + 1)
+ 3*Place (x, y)
- Place (x^3 + x + 1, y + 1)
```

```
>>> from sage.all import *
>>> K = FunctionField(GF(Integer(2)), names=('x',)); (x,) = K._first_ngens(1); R = K[
↳ 't']; (t,) = R._first_ngens(1)
>>> L = K.extension(t**Integer(3) + x**Integer(3)*t + x, names=('y',)); (y,) = L._
↳ first_ngens(1)
>>> f = x/(y+Integer(1))
>>> f.divisor()
- Place (1/x, 1/x^3*y^2 + 1/x)
+ Place (1/x, 1/x^3*y^2 + 1/x^2*y + 1)
+ 3*Place (x, y)
- Place (x^3 + x + 1, y + 1)
```

The Riemann-Roch space of a divisor can be computed. We can get a basis of the space as a vector space over the constant field:

```
sage: p = L.places_finite()[0]
sage: q = L.places_infinite()[0]
sage: (3*p + 2*q).basis_function_space()
[1/x*y^2 + x^2, 1, 1/x]
```

```
>>> from sage.all import *
>>> p = L.places_finite()[Integer(0)]
>>> q = L.places_infinite()[Integer(0)]
>>> (Integer(3)*p + Integer(2)*q).basis_function_space()
[1/x*y^2 + x^2, 1, 1/x]
```

We verify the Riemann-Roch theorem:

```
sage: D = 3*p - q
sage: index_of_speciality = len(D.basis_differential_space())
sage: D.dimension() == D.degree() - L.genus() + 1 + index_of_speciality
True
```

```
>>> from sage.all import *
>>> D = Integer(3)*p - q
>>> index_of_speciality = len(D.basis_differential_space())
>>> D.dimension() == D.degree() - L.genus() + Integer(1) + index_of_speciality
True
```

AUTHORS:

- Kwankyu Lee (2017-04-30): initial version

class sage.rings.function_field.divisor.**DivisorGroup**(*field*)

Bases: UniqueRepresentation, Parent

Groups of divisors of function fields.

INPUT:

- field – function field

EXAMPLES:

```
sage: K.<x> = FunctionField(GF(5)); _.<Y> = K[]
sage: F.<y> = K.extension(Y^2 - x^3 - 1)
sage: F.divisor_group()
Divisor group of Function field in y defined by y^2 + 4*x^3 + 4
```

```
>>> from sage.all import *
>>> K = FunctionField(GF(Integer(5)), names=('x',)); (x,) = K._first_ngens(1); _ =
↳ K['Y']; (Y,) = _._first_ngens(1)
>>> F = K.extension(Y**Integer(2) - x**Integer(3) - Integer(1), names=('y',)); (y,
↳ ) = F._first_ngens(1)
>>> F.divisor_group()
Divisor group of Function field in y defined by y^2 + 4*x^3 + 4
```

Element

alias of *FunctionFieldDivisor*

function_field()

Return the function field to which the divisor group is attached.

EXAMPLES:

```
sage: K.<x> = FunctionField(GF(5)); _.<Y> = K[]
sage: F.<y> = K.extension(Y^2 - x^3 - 1)
sage: G = F.divisor_group()
sage: G.function_field()
Function field in y defined by y^2 + 4*x^3 + 4
```

```
>>> from sage.all import *
>>> K = FunctionField(GF(Integer(5)), names=('x',)); (x,) = K._first_ngens(1);
(continues on next page)
```

(continued from previous page)

```

↪ _ = K['Y']; (Y,) = _._first_ngens(1)
>>> F = K.extension(Y**Integer(2) - x**Integer(3) - Integer(1), names=('y',));
↪ (y,) = F._first_ngens(1)
>>> G = F.divisor_group()
>>> G.function_field()
Function field in y defined by y^2 + 4*x^3 + 4

```

class sage.rings.function_field.divisor.**FunctionFieldDivisor**(parent, data)

Bases: `ModuleElement`

Divisors of function fields.

INPUT:

- parent – divisor group
- data – dictionary of place and multiplicity pairs

EXAMPLES:

```

sage: K.<x> = FunctionField(GF(2)); _.<Y> = K[]
sage: F.<y> = K.extension(Y^3 - x^2*(x^2 + x + 1)^2)
sage: f = x/(y + 1)
sage: f.divisor()
Place (1/x, 1/x^4*y^2 + 1/x^2*y + 1)
+ Place (1/x, 1/x^2*y + 1)
+ 3*Place (x, (1/(x^3 + x^2 + x))*y^2)
- 6*Place (x + 1, y + 1)

```

```

>>> from sage.all import *
>>> K = FunctionField(GF(Integer(2)), names=('x',)); (x,) = K._first_ngens(1); _
↪ = K['Y']; (Y,) = _._first_ngens(1)
>>> F = K.extension(Y**Integer(3) - x**Integer(2)*(x**Integer(2) + x +
↪ Integer(1))**Integer(2), names=('y',)); (y,) = F._first_ngens(1)
>>> f = x/(y + Integer(1))
>>> f.divisor()
Place (1/x, 1/x^4*y^2 + 1/x^2*y + 1)
+ Place (1/x, 1/x^2*y + 1)
+ 3*Place (x, (1/(x^3 + x^2 + x))*y^2)
- 6*Place (x + 1, y + 1)

```

basis_differential_space()

Return a basis of the space of differentials $\Omega(D)$ for the divisor D .

EXAMPLES:

We check the Riemann-Roch theorem:

```

sage: K.<x>=FunctionField(GF(4)); _.<Y> = K[]
sage: L.<y>=K.extension(Y^3 + x^3*Y + x)
sage: d = 3*L.places()[0]
sage: l = len(d.basis_function_space())
sage: i = len(d.basis_differential_space())
sage: l == d.degree() + 1 - L.genus() + i
True

```

```

>>> from sage.all import *
>>> K = FunctionField(GF(Integer(4)), names=('x',)); (x,) = K._first_ngens(1);
↪ _ = K['Y']; (Y,) = _._first_ngens(1)
>>> L = K.extension(Y**Integer(3) + x**Integer(3)*Y + x, names=('y',)); (y,) =
↪ L._first_ngens(1)
>>> d = Integer(3)*L.places()[Integer(0)]
>>> l = len(d.basis_function_space())
>>> i = len(d.basis_differential_space())
>>> l == d.degree() + Integer(1) - L.genus() + i
True

```

basis_function_space()

Return a basis of the Riemann-Roch space of the divisor.

EXAMPLES:

```

sage: K.<x> = FunctionField(GF(5)); _.<Y> = K[]
sage: F.<y> = K.extension(Y^2 - x^3 - 1)
sage: O = F.maximal_order()
sage: I = O.ideal(x - 2)
sage: D = I.divisor()
sage: D.basis_function_space()
[x/(x + 3), 1/(x + 3)]

```

```

>>> from sage.all import *
>>> K = FunctionField(GF(Integer(5)), names=('x',)); (x,) = K._first_ngens(1);
↪ _ = K['Y']; (Y,) = _._first_ngens(1)
>>> F = K.extension(Y**Integer(2) - x**Integer(3) - Integer(1), names=('y',));
↪ (y,) = F._first_ngens(1)
>>> O = F.maximal_order()
>>> I = O.ideal(x - Integer(2))
>>> D = I.divisor()
>>> D.basis_function_space()
[x/(x + 3), 1/(x + 3)]

```

degree()

Return the degree of the divisor.

EXAMPLES:

```

sage: K.<x> = FunctionField(GF(4)); _.<Y> = K[]
sage: L.<y> = K.extension(Y^3 + x^3*Y + x)
sage: p1,p2 = L.places()[2:]
sage: D = 2*p1 - 3*p2
sage: D.degree()
-1

```

```

>>> from sage.all import *
>>> K = FunctionField(GF(Integer(4)), names=('x',)); (x,) = K._first_ngens(1);
↪ _ = K['Y']; (Y,) = _._first_ngens(1)
>>> L = K.extension(Y**Integer(3) + x**Integer(3)*Y + x, names=('y',)); (y,) =
↪ L._first_ngens(1)
>>> p1,p2 = L.places()[Integer(2)]

```

(continues on next page)

(continued from previous page)

```
>>> D = Integer(2)*p1 - Integer(3)*p2
>>> D.degree()
-1
```

denominator()

Return the denominator part of the divisor.

The denominator of a divisor is the negative of the negative part of the divisor.

EXAMPLES:

```
sage: K.<x> = FunctionField(GF(4)); _.<Y> = K[]
sage: L.<y> = K.extension(Y^3 + x^3*Y + x)
sage: p1,p2 = L.places()[2:]
sage: D = 2*p1 - 3*p2
sage: D.denominator()
3*Place (1/x, 1/x^3*y^2 + 1/x^2*y + 1)
```

```
>>> from sage.all import *
>>> K = FunctionField(GF(Integer(4)), names=('x',)); (x,) = K._first_ngens(1);
↳ _ = K['Y']; (Y,) = _._first_ngens(1)
>>> L = K.extension(Y**Integer(3) + x**Integer(3)*Y + x, names=('y',)); (y,) =
↳ L._first_ngens(1)
>>> p1,p2 = L.places()[Integer(2):]
>>> D = Integer(2)*p1 - Integer(3)*p2
>>> D.denominator()
3*Place (1/x, 1/x^3*y^2 + 1/x^2*y + 1)
```

dict()

Return the dictionary representing the divisor.

EXAMPLES:

```
sage: K.<x> = FunctionField(GF(4)); _.<Y> = K[]
sage: L.<y> = K.extension(Y^3 + x^3*Y + x)
sage: f = x/(y + 1)
sage: D = f.divisor()
sage: D.dict()
{Place (1/x, 1/x^3*y^2 + 1/x): -1,
 Place (1/x, 1/x^3*y^2 + 1/x^2*y + 1): 1,
 Place (x, y): 3,
 Place (x^3 + x + 1, y + 1): -1}
```

```
>>> from sage.all import *
>>> K = FunctionField(GF(Integer(4)), names=('x',)); (x,) = K._first_ngens(1);
↳ _ = K['Y']; (Y,) = _._first_ngens(1)
>>> L = K.extension(Y**Integer(3) + x**Integer(3)*Y + x, names=('y',)); (y,) =
↳ L._first_ngens(1)
>>> f = x/(y + Integer(1))
>>> D = f.divisor()
>>> D.dict()
{Place (1/x, 1/x^3*y^2 + 1/x): -1,
 Place (1/x, 1/x^3*y^2 + 1/x^2*y + 1): 1,
```

(continues on next page)

(continued from previous page)

```
Place (x, y): 3,
Place (x^3 + x + 1, y + 1): -1}
```

differential_space()

Return the vector space of the differential space $\Omega(D)$ of the divisor D .

OUTPUT:

- a vector space isomorphic to $\Omega(D)$
- an isomorphism from the vector space to the differential space
- the inverse of the isomorphism

EXAMPLES:

```
sage: K.<x> = FunctionField(GF(5)); _.<Y> = K[]
sage: F.<y> = K.extension(Y^2 - x^3 - 1)
sage: O = F.maximal_order()
sage: I = O.ideal(x - 2)
sage: P1 = I.divisor().support()[0]
sage: Pinf = F.places_infinite()[0]
sage: D = -3*Pinf + P1
sage: V, from_V, to_V = D.differential_space()
sage: all(to_V(from_V(e)) == e for e in V)
True
```

```
>>> from sage.all import *
>>> K = FunctionField(GF(Integer(5)), names=('x',)); (x,) = K._first_ngens(1);
↳ _ = K['Y']; (Y,) = _._first_ngens(1)
>>> F = K.extension(Y**Integer(2) - x**Integer(3) - Integer(1), names=('y',));
↳ (y,) = F._first_ngens(1)
>>> O = F.maximal_order()
>>> I = O.ideal(x - Integer(2))
>>> P1 = I.divisor().support()[Integer(0)]
>>> Pinf = F.places_infinite()[Integer(0)]
>>> D = -Integer(3)*Pinf + P1
>>> V, from_V, to_V = D.differential_space()
>>> all(to_V(from_V(e)) == e for e in V)
True
```

dimension()

Return the dimension of the Riemann-Roch space of the divisor.

EXAMPLES:

```
sage: K.<x> = FunctionField(GF(5)); _.<Y> = K[]
sage: F.<y> = K.extension(Y^2 - x^3 - 1)
sage: O = F.maximal_order()
sage: I = O.ideal(x - 2)
sage: P1 = I.divisor().support()[0]
sage: Pinf = F.places_infinite()[0]
sage: D = 3*Pinf + 2*P1
sage: D.dimension()
5
```

```

>>> from sage.all import *
>>> K = FunctionField(GF(Integer(5)), names=('x',)); (x,) = K._first_ngens(1);
↪ _ = K['Y']; (Y,) = _._first_ngens(1)
>>> F = K.extension(Y**Integer(2) - x**Integer(3) - Integer(1), names=('y',));
↪ (y,) = F._first_ngens(1)
>>> O = F.maximal_order()
>>> I = O.ideal(x - Integer(2))
>>> P1 = I.divisor().support()[Integer(0)]
>>> Pinf = F.places_infinite()[Integer(0)]
>>> D = Integer(3)*Pinf + Integer(2)*P1
>>> D.dimension()
5

```

function_space()

Return the vector space of the Riemann-Roch space of the divisor.

OUTPUT:

- a vector space, an isomorphism from the vector space to the Riemann-Roch space, and its inverse.

EXAMPLES:

```

sage: K.<x> = FunctionField(GF(5)); _.<Y> = K[]
sage: F.<y> = K.extension(Y^2-x^3-1)
sage: O = F.maximal_order()
sage: I = O.ideal(x - 2)
sage: D = I.divisor()
sage: V, from_V, to_V = D.function_space()
sage: all(to_V(from_V(e)) == e for e in V)
True

```

```

>>> from sage.all import *
>>> K = FunctionField(GF(Integer(5)), names=('x',)); (x,) = K._first_ngens(1);
↪ _ = K['Y']; (Y,) = _._first_ngens(1)
>>> F = K.extension(Y**Integer(2)-x**Integer(3)-Integer(1), names=('y',)); (y,
↪ ) = F._first_ngens(1)
>>> O = F.maximal_order()
>>> I = O.ideal(x - Integer(2))
>>> D = I.divisor()
>>> V, from_V, to_V = D.function_space()
>>> all(to_V(from_V(e)) == e for e in V)
True

```

is_effective()

Return True if this divisor has nonnegative multiplicity at all places.

EXAMPLES:

```

sage: K.<x> = FunctionField(GF(4)); _.<Y> = K[]
sage: L.<y> = K.extension(Y^3 + x^3*Y + x)
sage: p1, p2 = L.places()[2:]
sage: D = 2*p1 + 3*p2
sage: D.is_effective()
True

```

(continues on next page)

(continued from previous page)

```
sage: E = D - 4*p2
sage: E.is_effective()
False
```

```
>>> from sage.all import *
>>> K = FunctionField(GF(Integer(4)), names=('x',)); (x,) = K._first_ngens(1);
↪ _ = K['Y']; (Y,) = _._first_ngens(1)
>>> L = K.extension(Y**Integer(3) + x**Integer(3)*Y + x, names=('y',)); (y,) =
↪ L._first_ngens(1)
>>> p1, p2 = L.places()[Integer(2)]
>>> D = Integer(2)*p1 + Integer(3)*p2
>>> D.is_effective()
True
>>> E = D - Integer(4)*p2
>>> E.is_effective()
False
```

list()

Return the list of place and multiplicity pairs of the divisor.

EXAMPLES:

```
sage: K.<x> = FunctionField(GF(4)); _.<Y> = K[]
sage: L.<y> = K.extension(Y^3 + x^3*Y + x)
sage: f = x/(y + 1)
sage: D = f.divisor()
sage: D.list()
[(Place (1/x, 1/x^3*y^2 + 1/x), -1),
 (Place (1/x, 1/x^3*y^2 + 1/x^2*y + 1), 1),
 (Place (x, y), 3),
 (Place (x^3 + x + 1, y + 1), -1)]
```

```
>>> from sage.all import *
>>> K = FunctionField(GF(Integer(4)), names=('x',)); (x,) = K._first_ngens(1);
↪ _ = K['Y']; (Y,) = _._first_ngens(1)
>>> L = K.extension(Y**Integer(3) + x**Integer(3)*Y + x, names=('y',)); (y,) =
↪ L._first_ngens(1)
>>> f = x/(y + Integer(1))
>>> D = f.divisor()
>>> D.list()
[(Place (1/x, 1/x^3*y^2 + 1/x), -1),
 (Place (1/x, 1/x^3*y^2 + 1/x^2*y + 1), 1),
 (Place (x, y), 3),
 (Place (x^3 + x + 1, y + 1), -1)]
```

multiplicity(*place*)

Return the multiplicity of the divisor at the place.

INPUT:

- *place* – place of a function field

EXAMPLES:


```

sage: K.<x> = FunctionField(GF(4)); _.<Y> = K[]
sage: L.<y> = K.extension(Y^3 + x^3*Y + x)
sage: p1,p2 = L.places()[2]
sage: D = 2*p1 - 3*p2
sage: D.multiplicity(p1)
2
sage: D.multiplicity(p2)
-3

```

```

>>> from sage.all import *
>>> K = FunctionField(GF(Integer(4)), names=('x',)); (x,) = K._first_ngens(1);
↪ _ = K['Y']; (Y,) = _._first_ngens(1)
>>> L = K.extension(Y**Integer(3) + x**Integer(3)*Y + x, names=('y',)); (y,) =
↪ L._first_ngens(1)
>>> p1,p2 = L.places()[Integer(2)]
>>> D = Integer(2)*p1 - Integer(3)*p2
>>> D.multiplicity(p1)
2
>>> D.multiplicity(p2)
-3

```

numerator()

Return the numerator part of the divisor.

The numerator of a divisor is the positive part of the divisor.

EXAMPLES:

```

sage: K.<x> = FunctionField(GF(4)); _.<Y> = K[]
sage: L.<y> = K.extension(Y^3 + x^3*Y + x)
sage: p1,p2 = L.places()[2]
sage: D = 2*p1 - 3*p2
sage: D.numerator()
2*Place (1/x, 1/x^3*y^2 + 1/x)

```

```

>>> from sage.all import *
>>> K = FunctionField(GF(Integer(4)), names=('x',)); (x,) = K._first_ngens(1);
↪ _ = K['Y']; (Y,) = _._first_ngens(1)
>>> L = K.extension(Y**Integer(3) + x**Integer(3)*Y + x, names=('y',)); (y,) =
↪ L._first_ngens(1)
>>> p1,p2 = L.places()[Integer(2)]
>>> D = Integer(2)*p1 - Integer(3)*p2
>>> D.numerator()
2*Place (1/x, 1/x^3*y^2 + 1/x)

```

support()

Return the support of the divisor.

EXAMPLES:

```

sage: K.<x> = FunctionField(GF(4)); _.<Y> = K[]
sage: L.<y> = K.extension(Y^3 + x^3*Y + x)
sage: f = x/(y + 1)
sage: D = f.divisor()

```

(continues on next page)

(continued from previous page)

```
sage: D.support()
[Place (1/x, 1/x^3*y^2 + 1/x),
 Place (1/x, 1/x^3*y^2 + 1/x^2*y + 1),
 Place (x, y),
 Place (x^3 + x + 1, y + 1)]
```

```
>>> from sage.all import *
>>> K = FunctionField(GF(Integer(4)), names=('x',)); (x,) = K._first_ngens(1);
↪ _ = K['Y']; (Y,) = _._first_ngens(1)
>>> L = K.extension(Y**Integer(3) + x**Integer(3)*Y + x, names=('y',)); (y,) =
↪ L._first_ngens(1)
>>> f = x/(y + Integer(1))
>>> D = f.divisor()
>>> D.support()
[Place (1/x, 1/x^3*y^2 + 1/x),
 Place (1/x, 1/x^3*y^2 + 1/x^2*y + 1),
 Place (x, y),
 Place (x^3 + x + 1, y + 1)]
```

valuation (*place*)

Return the multiplicity of the divisor at the place.

INPUT:

- *place* – place of a function field

EXAMPLES:

```
sage: K.<x> = FunctionField(GF(4)); _.<Y> = K[]
sage: L.<y> = K.extension(Y^3 + x^3*Y + x)
sage: p1,p2 = L.places()[ :2]
sage: D = 2*p1 - 3*p2
sage: D.multiplicity(p1)
2
sage: D.multiplicity(p2)
-3
```

```
>>> from sage.all import *
>>> K = FunctionField(GF(Integer(4)), names=('x',)); (x,) = K._first_ngens(1);
↪ _ = K['Y']; (Y,) = _._first_ngens(1)
>>> L = K.extension(Y**Integer(3) + x**Integer(3)*Y + x, names=('y',)); (y,) =
↪ L._first_ngens(1)
>>> p1,p2 = L.places()[ :Integer(2)]
>>> D = Integer(2)*p1 - Integer(3)*p2
>>> D.multiplicity(p1)
2
>>> D.multiplicity(p2)
-3
```

`sage.rings.function_field.divisor.divisor` (*field*, *data*)

Construct a divisor from the data.

INPUT:

- *field* – function field

- data – dictionary of place and multiplicity pairs

EXAMPLES:

```
sage: K.<x> = FunctionField(GF(2)); R.<t> = K[]
sage: F.<y> = K.extension(t^3 - x^2*(x^2 + x + 1)^2)
sage: from sage.rings.function_field.divisor import divisor
sage: p, q, r = F.places()
sage: divisor(F, {p: 1, q: 2, r: 3})
Place (1/x, 1/x^2*y + 1)
+ 2*Place (x, (1/(x^3 + x^2 + x))*y^2)
+ 3*Place (x + 1, y + 1)
```

```
>>> from sage.all import *
>>> K = FunctionField(GF(Integer(2)), names=('x',)); (x,) = K._first_ngens(1); R_
↳ K['t']; (t,) = R._first_ngens(1)
>>> F = K.extension(t**Integer(3) - x**Integer(2)*(x**Integer(2) + x +
↳ Integer(1))**Integer(2), names=('y',)); (y,) = F._first_ngens(1)
>>> from sage.rings.function_field.divisor import divisor
>>> p, q, r = F.places()
>>> divisor(F, {p: Integer(1), q: Integer(2), r: Integer(3)})
Place (1/x, 1/x^2*y + 1)
+ 2*Place (x, (1/(x^3 + x^2 + x))*y^2)
+ 3*Place (x + 1, y + 1)
```

sage.rings.function_field.divisor.**prime_divisor** (*field, place, m=1*)

Construct a prime divisor from the place.

INPUT:

- field – function field
- place – place of the function field
- m – (default: 1) a positive integer; multiplicity at the place

EXAMPLES:

```
sage: K.<x> = FunctionField(GF(2)); R.<t> = K[]
sage: F.<y> = K.extension(t^3 - x^2*(x^2 + x + 1)^2)
sage: p = F.places()[0]
sage: from sage.rings.function_field.divisor import prime_divisor
sage: d = prime_divisor(F, p)
sage: 3 * d == prime_divisor(F, p, 3)
True
```

```
>>> from sage.all import *
>>> K = FunctionField(GF(Integer(2)), names=('x',)); (x,) = K._first_ngens(1); R_
↳ K['t']; (t,) = R._first_ngens(1)
>>> F = K.extension(t**Integer(3) - x**Integer(2)*(x**Integer(2) + x +
↳ Integer(1))**Integer(2), names=('y',)); (y,) = F._first_ngens(1)
>>> p = F.places()[Integer(0)]
>>> from sage.rings.function_field.divisor import prime_divisor
>>> d = prime_divisor(F, p)
>>> Integer(3) * d == prime_divisor(F, p, Integer(3))
True
```


DIFFERENTIALS OF FUNCTION FIELDS

Sage provides arithmetic with differentials of function fields.

EXAMPLES:

The module of differentials on a function field forms an one-dimensional vector space over the function field:

```
sage: # needs sage.rings.finite_rings sage.rings.function_field
sage: K.<x> = FunctionField(GF(4)); _.<Y> = K[]
sage: L.<y> = K.extension(Y^3 + x + x^3*Y)
sage: f = x + y
sage: g = 1 / y
sage: df = f.differential()
sage: dg = g.differential()
sage: dfdg = f.derivative() / g.derivative()
sage: df == dfdg * dg
True
sage: df
(x*y^2 + 1/x*y + 1) d(x)
sage: df.parent()
Space of differentials of Function field in y defined by y^3 + x^3*y + x
```

```
>>> from sage.all import *
>>> # needs sage.rings.finite_rings sage.rings.function_field
>>> K = FunctionField(GF(Integer(4)), names=('x',)); (x,) = K._first_ngens(1); _ = K[
↳ 'Y']; (Y,) = _._first_ngens(1)
>>> L = K.extension(Y**Integer(3) + x + x**Integer(3)*Y, names=('y',)); (y,) = L._
↳ first_ngens(1)
>>> f = x + y
>>> g = Integer(1) / y
>>> df = f.differential()
>>> dg = g.differential()
>>> dfdg = f.derivative() / g.derivative()
>>> df == dfdg * dg
True
>>> df
(x*y^2 + 1/x*y + 1) d(x)
>>> df.parent()
Space of differentials of Function field in y defined by y^3 + x^3*y + x
```

We can compute a canonical divisor:

```
sage: # needs sage.rings.finite_rings sage.rings.function_field
sage: k = df.divisor()
sage: k.degree()
4
sage: k.degree() == 2 * L.genus() - 2
True
```

```
>>> from sage.all import *
>>> # needs sage.rings.finite_rings sage.rings.function_field
>>> k = df.divisor()
>>> k.degree()
4
>>> k.degree() == Integer(2) * L.genus() - Integer(2)
True
```

Exact differentials vanish and logarithmic differentials are stable under the Cartier operation:

```
sage: # needs sage.rings.finite_rings sage.rings.function_field
sage: df.cartier()
0
sage: w = 1/f * df
sage: w.cartier() == w
True
```

```
>>> from sage.all import *
>>> # needs sage.rings.finite_rings sage.rings.function_field
>>> df.cartier()
0
>>> w = Integer(1)/f * df
>>> w.cartier() == w
True
```

AUTHORS:

- Kwankyu Lee (2017-04-30): initial version

class sage.rings.function_field.differential.DifferentialSpace (*field*, *category=None*)

Bases: UniqueRepresentation, Parent

Space of differentials of a function field.

INPUT:

- *field* – function field

EXAMPLES:

```
sage: K.<x> = FunctionField(GF(4)); _.<Y> = K[] #_
↪needs sage.rings.finite_rings
sage: L.<y> = K.extension(Y^3 + x^3*Y + x) #_
↪needs sage.rings.finite_rings sage.rings.function_field
sage: L.space_of_differentials() #_
↪needs sage.rings.finite_rings sage.rings.function_field
Space of differentials of Function field in y defined by y^3 + x^3*y + x
```

```

>>> from sage.all import *
>>> K = FunctionField(GF(Integer(4)), names=('x',)); (x,) = K._first_ngens(1); _
↳ K['Y']; (Y,) = _._first_ngens(1) # needs sage.rings.finite_rings
>>> L = K.extension(Y**Integer(3) + x**Integer(3)*Y + x, names=('y',)); (y,) = L._
↳ first_ngens(1) # needs sage.rings.finite_rings sage.rings.function_field
>>> L.space_of_differentials() #
↳ needs sage.rings.finite_rings sage.rings.function_field
Space of differentials of Function field in y defined by  $y^3 + x^3y + x$ 

```

The space of differentials is a one-dimensional module over the function field. So a base differential is chosen to represent differentials. Usually the generator of the base rational function field is a separating element and used to generate the base differential. Otherwise a separating element is automatically found and used to generate the base differential relative to which other differentials are denoted:

```

sage: # needs sage.rings.function_field
sage: K.<x> = FunctionField(GF(5))
sage: R.<y> = K[]
sage: L.<y> = K.extension(y^5 - 1/x)
sage: L(x).differential()
0
sage: y.differential()
d(y)
sage: (y^2).differential()
(2*y) d(y)

```

```

>>> from sage.all import *
>>> # needs sage.rings.function_field
>>> K = FunctionField(GF(Integer(5)), names=('x',)); (x,) = K._first_ngens(1)
>>> R = K['y']; (y,) = R._first_ngens(1)
>>> L = K.extension(y**Integer(5) - Integer(1)/x, names=('y',)); (y,) = L._first_
↳ ngens(1)
>>> L(x).differential()
0
>>> y.differential()
d(y)
>>> (y**Integer(2)).differential()
(2*y) d(y)

```

Element

alias of *FunctionFieldDifferential*

basis()

Return a basis.

EXAMPLES:

```

sage: # needs sage.rings.finite_rings sage.rings.function_field
sage: K.<x> = FunctionField(GF(4)); _.<Y> = K[]
sage: L.<y> = K.extension(Y^3 + x^3*Y + x)
sage: S = L.space_of_differentials()
sage: S.basis()
Family (d(x),)

```

```

>>> from sage.all import *
>>> # needs sage.rings.finite_rings sage.rings.function_field
>>> K = FunctionField(GF(Integer(4)), names=('x',)); (x,) = K._first_ngens(1);
↳ _ = K['Y']; (Y,) = _._first_ngens(1)
>>> L = K.extension(Y**Integer(3) + x**Integer(3)*Y + x, names=('y',)); (y,) =
↳ L._first_ngens(1)
>>> S = L.space_of_differentials()
>>> S.basis()
Family (d(x),)

```

function_field()

Return the function field to which the space of differentials is attached.

EXAMPLES:

```

sage: # needs sage.rings.finite_rings sage.rings.function_field
sage: K.<x> = FunctionField(GF(4)); _.<Y> = K[]
sage: L.<y> = K.extension(Y^3 + x^3*Y + x)
sage: S = L.space_of_differentials()
sage: S.function_field()
Function field in y defined by y^3 + x^3*y + x

```

```

>>> from sage.all import *
>>> # needs sage.rings.finite_rings sage.rings.function_field
>>> K = FunctionField(GF(Integer(4)), names=('x',)); (x,) = K._first_ngens(1);
↳ _ = K['Y']; (Y,) = _._first_ngens(1)
>>> L = K.extension(Y**Integer(3) + x**Integer(3)*Y + x, names=('y',)); (y,) =
↳ L._first_ngens(1)
>>> S = L.space_of_differentials()
>>> S.function_field()
Function field in y defined by y^3 + x^3*y + x

```

class sage.rings.function_field.differential.DifferentialsSpaceInclusion

Bases: `Morphism`

Inclusion morphisms for extensions of function fields.

EXAMPLES:

```

sage: K.<x> = FunctionField(QQ); R.<y> = K[]
sage: L.<y> = K.extension(y^2 - x*y + 4*x^3)
↳ needs sage.rings.function_field
sage: OK = K.space_of_differentials()
sage: OL = L.space_of_differentials()
↳ needs sage.rings.function_field
sage: OL.coerce_map_from(OK)
↳ needs sage.rings.function_field
Inclusion morphism:
From: Space of differentials of Rational function field in x over Rational Field
To: Space of differentials of Function field in y defined by y^2 - x*y + 4*x^3

```

```

>>> from sage.all import *
>>> K = FunctionField(QQ, names=('x',)); (x,) = K._first_ngens(1); R = K['y']; (y,
↳ ) = R._first_ngens(1)

```

(continues on next page)

(continued from previous page)

```

>>> L = K.extension(y**Integer(2) - x*y + Integer(4)*x**Integer(3), names=('y',));
↳ (y,) = L._first_ngens(1) # needs sage.rings.function_field
>>> OK = K.space_of_differentials()
>>> OL = L.space_of_differentials() #_
↳ needs sage.rings.function_field
>>> OL.coerce_map_from(OK) #_
↳ needs sage.rings.function_field
Inclusion morphism:
  From: Space of differentials of Rational function field in x over Rational Field
  To:   Space of differentials of Function field in y defined by  $y^2 - xy + 4x^3$ 

```

is_injective()

Return True, since the inclusion morphism is injective.

EXAMPLES:

```

sage: K.<x> = FunctionField(QQ); R.<y> = K[]
sage: L.<y> = K.extension(y^2 - x*y + 4*x^3) #_
↳ needs sage.rings.function_field
sage: OK = K.space_of_differentials()
sage: OL = L.space_of_differentials() #_
↳ needs sage.rings.function_field
sage: OL.coerce_map_from(OK).is_injective() #_
↳ needs sage.rings.function_field
True

```

```

>>> from sage.all import *
>>> K = FunctionField(QQ, names=('x',)); (x,) = K._first_ngens(1); R = K['y'];
↳ (y,) = R._first_ngens(1)
>>> L = K.extension(y**Integer(2) - x*y + Integer(4)*x**Integer(3), names=('y',)); (y,) = L._first_ngens(1) # needs sage.rings.function_field
>>> OK = K.space_of_differentials()
>>> OL = L.space_of_differentials() #_
↳ needs sage.rings.function_field
>>> OL.coerce_map_from(OK).is_injective() #_
↳ needs sage.rings.function_field
True

```

is_surjective()

Return True if the inclusion morphism is surjective.

EXAMPLES:

```

sage: # needs sage.rings.function_field
sage: K.<x> = FunctionField(QQ); R.<y> = K[]
sage: L.<y> = K.extension(y^2 - x*y + 4*x^3)
sage: OK = K.space_of_differentials()
sage: OL = L.space_of_differentials()
sage: OL.coerce_map_from(OK).is_surjective()
False
sage: S.<z> = L[]
sage: M.<z> = L.extension(z - 1)
sage: OM = M.space_of_differentials()

```

(continues on next page)

(continued from previous page)

```
sage: OM.coerce_map_from(OL).is_surjective()
True
```

```
>>> from sage.all import *
>>> # needs sage.rings.function_field
>>> K = FunctionField(QQ, names=('x',)); (x,) = K._first_ngens(1); R = K['y'];
↪ (y,) = R._first_ngens(1)
>>> L = K.extension(y**Integer(2) - x*y + Integer(4)*x**Integer(3), names=('y
↪ ',)); (y,) = L._first_ngens(1)
>>> OK = K.space_of_differentials()
>>> OL = L.space_of_differentials()
>>> OL.coerce_map_from(OK).is_surjective()
False
>>> S = L['z']; (z,) = S._first_ngens(1)
>>> M = L.extension(z - Integer(1), names=('z',)); (z,) = M._first_ngens(1)
>>> OM = M.space_of_differentials()
>>> OM.coerce_map_from(OL).is_surjective()
True
```

```
class sage.rings.function_field.differential.DifferentialsSpace_global (field,
                                                                    category=None)
```

Bases: *DifferentialsSpace*

Space of differentials of a global function field.

INPUT:

- field – function field

EXAMPLES:

```
sage: K.<x> = FunctionField(GF(4)); _.<Y> = K[] #_
↪ needs sage.rings.finite_rings
sage: L.<y> = K.extension(Y^3 + x^3*Y + x) #_
↪ needs sage.rings.finite_rings sage.rings.function_field
sage: L.space_of_differentials() #_
↪ needs sage.rings.finite_rings sage.rings.function_field
Space of differentials of Function field in y defined by y^3 + x^3*y + x
```

```
>>> from sage.all import *
>>> K = FunctionField(GF(Integer(4)), names=('x',)); (x,) = K._first_ngens(1); _
↪ = K['Y']; (Y,) = _.first_ngens(1) # needs sage.rings.finite_rings
>>> L = K.extension(Y**Integer(3) + x**Integer(3)*Y + x, names=('y',)); (y,) = L._
↪ first_ngens(1) # needs sage.rings.finite_rings sage.rings.function_field
>>> L.space_of_differentials() #_
↪ needs sage.rings.finite_rings sage.rings.function_field
Space of differentials of Function field in y defined by y^3 + x^3*y + x
```

Element

alias of *FunctionFieldDifferential_global*

```
class sage.rings.function_field.differential.FunctionFieldDifferential (parent, f, t=None)
```

Bases: *ModuleElement*

Base class for differentials on function fields.

INPUT:

- f – element of the function field
- t – element of the function field; if t is not specified, the generator of the base differential is assumed

EXAMPLES:

```
sage: F.<x> = FunctionField(QQ)
sage: f = x/(x^2 + x + 1)
sage: f.differential()
((-x^2 + 1)/(x^4 + 2*x^3 + 3*x^2 + 2*x + 1)) d(x)
```

```
>>> from sage.all import *
>>> F = FunctionField(QQ, names=('x',)); (x,) = F._first_ngens(1)
>>> f = x/(x**Integer(2) + x + Integer(1))
>>> f.differential()
((-x^2 + 1)/(x^4 + 2*x^3 + 3*x^2 + 2*x + 1)) d(x)
```

```
sage: K.<x> = FunctionField(QQ); _.<Y> = K[]
sage: L.<y> = K.extension(Y^3 + x + x^3*Y) #_
↳needs sage.rings.function_field
sage: L(x).differential() #_
↳needs sage.rings.function_field
d(x)
sage: y.differential() #_
↳needs sage.rings.function_field
((21/4*x/(x^7 + 27/4))*y^2 + ((3/2*x^7 + 9/4)/(x^8 + 27/4*x))*y + 7/2*x^4/(x^7 +
↳27/4)) d(x)
```

```
>>> from sage.all import *
>>> K = FunctionField(QQ, names=('x',)); (x,) = K._first_ngens(1); _ = K['Y']; (Y,
↳) = _.first_ngens(1)
>>> L = K.extension(Y**Integer(3) + x + x**Integer(3)*Y, names=('y',)); (y,) = L.
↳first_ngens(1) # needs sage.rings.function_field
>>> L(x).differential() #_
↳needs sage.rings.function_field
d(x)
>>> y.differential() #_
↳needs sage.rings.function_field
((21/4*x/(x^7 + 27/4))*y^2 + ((3/2*x^7 + 9/4)/(x^8 + 27/4*x))*y + 7/2*x^4/(x^7 +
↳27/4)) d(x)
```

divisor()

Return the divisor of the differential.

EXAMPLES:

```
sage: K.<x> = FunctionField(GF(5)); _.<Y> = K[]
sage: L.<y> = K.extension(Y^3 + x + x^3*Y) #_
↳needs sage.rings.function_field
sage: w = (1/y) * y.differential() #_
↳needs sage.rings.function_field
sage: w.divisor() #_
↳needs sage.rings.function_field
```

(continues on next page)

(continued from previous page)

```

- Place (1/x, 1/x^3*y^2 + 1/x)
- Place (1/x, 1/x^3*y^2 + 1/x^2*y + 1)
- Place (x, y)
+ Place (x + 2, y + 3)
+ Place (x^6 + 3*x^5 + 4*x^4 + 2*x^3 + x^2 + 3*x + 4, y + x^5)

```

```

>>> from sage.all import *
>>> K = FunctionField(GF(Integer(5)), names=('x',)); (x,) = K._first_ngens(1);
↳ _ = K['Y']; (Y,) = _._first_ngens(1)
>>> L = K.extension(Y**Integer(3) + x + x**Integer(3)*Y, names=('y',)); (y,) = L._first_ngens(1) # needs sage.rings.function_field
↳ L._first_ngens(1) # needs sage.rings.function_field
>>> w = (Integer(1)/y) * y.differential()
↳ # needs sage.rings.function_field
>>> w.divisor()
↳ #
↳ needs sage.rings.function_field
- Place (1/x, 1/x^3*y^2 + 1/x)
- Place (1/x, 1/x^3*y^2 + 1/x^2*y + 1)
- Place (x, y)
+ Place (x + 2, y + 3)
+ Place (x^6 + 3*x^5 + 4*x^4 + 2*x^3 + x^2 + 3*x + 4, y + x^5)

```

```

sage: F.<x> = FunctionField(QQ)
sage: w = (1/x).differential()
sage: w.divisor()
↳ #
↳ needs sage.libs.pari
-2*Place (x)

```

```

>>> from sage.all import *
>>> F = FunctionField(QQ, names=('x',)); (x,) = F._first_ngens(1)
>>> w = (Integer(1)/x).differential()
>>> w.divisor()
↳ #
↳ needs sage.libs.pari
-2*Place (x)

```

monomial_coefficients (*copy=True*)

Return a dictionary whose keys are indices of basis elements in the support of `self` and whose values are the corresponding coefficients.

EXAMPLES:

```

sage: K.<x> = FunctionField(GF(5)); _.<Y> = K[]
sage: L.<y> = K.extension(Y^3 + x + x^3*Y)
↳ needs sage.rings.function_field
sage: d = y.differential()
↳ needs sage.rings.function_field
sage: d
↳ needs sage.rings.function_field
((4*x/(x^7 + 3))*y^2 + ((4*x^7 + 1)/(x^8 + 3*x))*y + x^4/(x^7 + 3)) d(x)
sage: d.monomial_coefficients()
↳ #
↳ needs sage.rings.function_field
{0: (4*x/(x^7 + 3))*y^2 + ((4*x^7 + 1)/(x^8 + 3*x))*y + x^4/(x^7 + 3)}

```

```

>>> from sage.all import *
>>> K = FunctionField(GF(Integer(5)), names=('x',)); (x,) = K._first_ngens(1);
↪ _ = K['Y']; (Y,) = _._first_ngens(1)
>>> L = K.extension(Y**Integer(3) + x + x**Integer(3)*Y, names=('y',)); (y,) =
↪ L._first_ngens(1) # needs sage.rings.function_field
>>> d = y.differential() #
↪ needs sage.rings.function_field
>>> d #
↪ needs sage.rings.function_field
((4*x/(x^7 + 3))*y^2 + ((4*x^7 + 1)/(x^8 + 3*x))*y + x^4/(x^7 + 3)) d(x)
>>> d.monomial_coefficients() #
↪ needs sage.rings.function_field
{0: (4*x/(x^7 + 3))*y^2 + ((4*x^7 + 1)/(x^8 + 3*x))*y + x^4/(x^7 + 3)}

```

residue (place)

Return the residue of the differential at the place.

INPUT:

- place – a place of the function field

OUTPUT: an element of the residue field of the place

EXAMPLES:

We verify the residue theorem in a rational function field:

```

sage: # needs sage.rings.finite_rings
sage: F.<x> = FunctionField(GF(4))
sage: f = 0
sage: while f == 0:
....:     f = F.random_element()
sage: w = 1/f * f.differential()
sage: d = f.divisor()
sage: s = d.support()
sage: sum([w.residue(p).trace() for p in s]) #
↪ needs sage.rings.function_field
0

```

```

>>> from sage.all import *
>>> # needs sage.rings.finite_rings
>>> F = FunctionField(GF(Integer(4)), names=('x',)); (x,) = F._first_ngens(1)
>>> f = Integer(0)
>>> while f == Integer(0):
...     f = F.random_element()
>>> w = Integer(1)/f * f.differential()
>>> d = f.divisor()
>>> s = d.support()
>>> sum([w.residue(p).trace() for p in s]) #
↪ needs sage.rings.function_field
0

```

and in an extension field:

```

sage: # needs sage.rings.function_field
sage: K.<x> = FunctionField(GF(7)); _.<Y> = K[]
sage: L.<y> = K.extension(Y^3 + x + x^3*Y)
sage: f = 0
sage: while f == 0:
....:     f = L.random_element()
sage: w = 1/f * f.differential()
sage: d = f.divisor()
sage: s = d.support()
sage: sum([w.residue(p).trace() for p in s])
0

```

```

>>> from sage.all import *
>>> # needs sage.rings.function_field
>>> K = FunctionField(GF(Integer(7)), names=('x',)); (x,) = K._first_ngens(1)
↪ _ = K['Y']; (Y,) = _._first_ngens(1)
>>> L = K.extension(Y**Integer(3) + x + x**Integer(3)*Y, names=('y',)); (y,) = L._first_ngens(1)
>>> f = Integer(0)
>>> while f == Integer(0):
...     f = L.random_element()
>>> w = Integer(1)/f * f.differential()
>>> d = f.divisor()
>>> s = d.support()
>>> sum([w.residue(p).trace() for p in s])
0

```

and also in a function field of characteristic zero:

```

sage: # needs sage.rings.function_field
sage: R.<x> = FunctionField(QQ)
sage: L.<Y> = R[]
sage: F.<y> = R.extension(Y^2 - x^4 - 4*x^3 - 2*x^2 - 1)
sage: a = 6*x^2 + 5*x + 7
sage: b = 2*x^6 + 8*x^5 + 3*x^4 - 4*x^3 - 1
sage: w = y*a/b*x.differential()
sage: d = w.divisor()
sage: sum([QQ(w.residue(p)) for p in d.support()])
0

```

```

>>> from sage.all import *
>>> # needs sage.rings.function_field
>>> R = FunctionField(QQ, names=('x',)); (x,) = R._first_ngens(1)
>>> L = R['Y']; (Y,) = L._first_ngens(1)
>>> F = R.extension(Y**Integer(2) - x**Integer(4) - Integer(4)*x**Integer(3) -
↪ Integer(2)*x**Integer(2) - Integer(1), names=('y',)); (y,) = F._first_
↪ ngens(1)
>>> a = Integer(6)*x**Integer(2) + Integer(5)*x + Integer(7)
>>> b = Integer(2)*x**Integer(6) + Integer(8)*x**Integer(5) +
↪ Integer(3)*x**Integer(4) - Integer(4)*x**Integer(3) - Integer(1)
>>> w = y*a/b*x.differential()
>>> d = w.divisor()

```

(continues on next page)

(continued from previous page)

```
>>> sum([QQ(w.residue(p)) for p in d.support()])
0
```

valuation (*place*)

Return the valuation of the differential at the place.

INPUT:

- *place* – a place of the function field

EXAMPLES:

```
sage: K.<x> = FunctionField(GF(5)); _.<Y> = K[]
sage: L.<y> = K.extension(Y^3 + x + x^3*Y) #_
↳needs sage.rings.function_field
sage: w = (1/y) * y.differential() #_
↳needs sage.rings.function_field
sage: [w.valuation(p) for p in L.places()] #_
↳needs sage.rings.function_field
[-1, -1, -1, 0, 1, 0]
```

```
>>> from sage.all import *
>>> K = FunctionField(GF(Integer(5)), names=('x',)); (x,) = K._first_ngens(1);
↳ _ = K['Y']; (Y,) = _._first_ngens(1)
>>> L = K.extension(Y**Integer(3) + x + x**Integer(3)*Y, names=('y',)); (y,) #_
↳ L._first_ngens(1) # needs sage.rings.function_field
>>> w = (Integer(1)/y) * y.differential() #_
↳ # needs sage.rings.function_field
>>> [w.valuation(p) for p in L.places()] #_
↳needs sage.rings.function_field
[-1, -1, -1, 0, 1, 0]
```

class sage.rings.function_field.differential.**FunctionFieldDifferential_global** (*parent, f, t=None*)

Bases: *FunctionFieldDifferential*

Differentials on global function fields.

EXAMPLES:

```
sage: F.<x> = FunctionField(GF(7))
sage: f = x/(x^2 + x + 1)
sage: f.differential()
((6*x^2 + 1)/(x^4 + 2*x^3 + 3*x^2 + 2*x + 1)) d(x)
```

```
>>> from sage.all import *
>>> F = FunctionField(GF(Integer(7)), names=('x',)); (x,) = F._first_ngens(1)
>>> f = x/(x**Integer(2) + x + Integer(1))
>>> f.differential()
((6*x^2 + 1)/(x^4 + 2*x^3 + 3*x^2 + 2*x + 1)) d(x)
```

```
sage: K.<x> = FunctionField(GF(4)); _.<Y> = K[] #_
↳needs sage.rings.finite_rings
```

(continues on next page)

(continued from previous page)

```
sage: L.<y> = K.extension(Y^3 + x + x^3*Y) #_
↪needs sage.rings.finite_rings sage.rings.function_field
sage: y.differential() #_
↪needs sage.rings.finite_rings sage.rings.function_field
(x*y^2 + 1/x*y) d(x)
```

```
>>> from sage.all import *
>>> K = FunctionField(GF(Integer(4)), names=('x',)); (x,) = K._first_ngens(1); _
↪= K['Y']; (Y,) = _._first_ngens(1) # needs sage.rings.finite_rings
>>> L = K.extension(Y**Integer(3) + x + x**Integer(3)*Y, names=('y',)); (y,) = L._
↪first_ngens(1) # needs sage.rings.finite_rings sage.rings.function_field
>>> y.differential() #_
↪needs sage.rings.finite_rings sage.rings.function_field
(x*y^2 + 1/x*y) d(x)
```

cartier()

Return the image of the differential by the Cartier operator.

The Cartier operator operates on differentials. Let x be a separating element of the function field. If a differential ω is written in prime-power representation $\omega = (f_0^p + f_1^p x + \cdots + f_{p-1}^p x^{p-1})dx$, then the Cartier operator maps ω to $f_{p-1}dx$. It is known that this definition does not depend on the choice of x .

The Cartier operator has interesting properties. Notably, the set of exact differentials is precisely the kernel of the Cartier operator and logarithmic differentials are stable under the Cartier operation.

EXAMPLES:

```
sage: # needs sage.rings.finite_rings sage.rings.function_field
sage: K.<x> = FunctionField(GF(4)); _.<Y> = K[]
sage: L.<y> = K.extension(Y^3 + x + x^3*Y)
sage: f = x/y
sage: w = 1/f*f.differential()
sage: w.cartier() == w
True
```

```
>>> from sage.all import *
>>> # needs sage.rings.finite_rings sage.rings.function_field
>>> K = FunctionField(GF(Integer(4)), names=('x',)); (x,) = K._first_ngens(1);
↪ _ = K['Y']; (Y,) = _._first_ngens(1)
>>> L = K.extension(Y**Integer(3) + x + x**Integer(3)*Y, names=('y',)); (y,)
↪ = L._first_ngens(1)
>>> f = x/y
>>> w = Integer(1)/f*f.differential()
>>> w.cartier() == w
True
```

```
sage: # needs sage.rings.finite_rings
sage: F.<x> = FunctionField(GF(4))
sage: f = x/(x^2 + x + 1)
sage: w = 1/f*f.differential()
sage: w.cartier() == w #_
↪needs sage.rings.function_field
True
```



```
>>> from sage.all import *
>>> # needs sage.rings.finite_rings
>>> F = FunctionField(GF(Integer(4)), names=('x',)); (x,) = F._first_ngens(1)
>>> f = x/(x**Integer(2) + x + Integer(1))
>>> w = Integer(1)/f*f.differential()
>>> w.cartier() == w #_
↪needs sage.rings.function_field
True
```


VALUATION RINGS OF FUNCTION FIELDS

A valuation ring of a function field is associated with a place of the function field. The valuation ring consists of all elements of the function field that have nonnegative valuation at the place.

EXAMPLES:

```
sage: K.<x> = FunctionField(GF(2)); _.<Y> = K[]
sage: L.<y> = K.extension(Y^2 + Y + x + 1/x)
sage: p = L.places_finite()[0]
sage: p
Place (x, x*y)
sage: R = p.valuation_ring()
sage: R
Valuation ring at Place (x, x*y)
sage: R.place() == p
True
```

```
>>> from sage.all import *
>>> K = FunctionField(GF(Integer(2)), names=('x',)); (x,) = K._first_ngens(1); _ = K[
↳ 'Y']; (Y,) = _._first_ngens(1)
>>> L = K.extension(Y**Integer(2) + Y + x + Integer(1)/x, names=('y',)); (y,) = L._
↳ first_ngens(1)
>>> p = L.places_finite()[Integer(0)]
>>> p
Place (x, x*y)
>>> R = p.valuation_ring()
>>> R
Valuation ring at Place (x, x*y)
>>> R.place() == p
True
```

Thus any nonzero element or its inverse of the function field lies in the valuation ring, as shown in the following example:

```
sage: f = y/(1+y)
sage: f in R
True
sage: f not in R
False
sage: f.valuation(p)
0
```

```
>>> from sage.all import *
>>> f = y/(Integer(1)+y)
>>> f in R
True
>>> f not in R
False
>>> f.valuation(p)
0
```

The residue field at the place is defined as the quotient ring of the valuation ring modulo its unique maximal ideal. The method `residue_field()` of the valuation ring returns an extension field of the constant base field, isomorphic to the residue field, along with lifting and evaluation homomorphisms:

```
sage: k,phi,psi = R.residue_field()
sage: k
Finite Field of size 2
sage: phi
Ring morphism:
  From: Finite Field of size 2
  To:   Valuation ring at Place (x, x*y)
sage: psi
Ring morphism:
  From: Valuation ring at Place (x, x*y)
  To:   Finite Field of size 2
sage: psi(f) in k
True
```

```
>>> from sage.all import *
>>> k,phi,psi = R.residue_field()
>>> k
Finite Field of size 2
>>> phi
Ring morphism:
  From: Finite Field of size 2
  To:   Valuation ring at Place (x, x*y)
>>> psi
Ring morphism:
  From: Valuation ring at Place (x, x*y)
  To:   Finite Field of size 2
>>> psi(f) in k
True
```

AUTHORS:

- Kwankyu Lee (2017-04-30): initial version

```
class sage.rings.function_field.valuation_ring.FunctionFieldValuationRing(field, place,
                                                                           category=None)
```

Bases: `UniqueRepresentation, Parent`

Base class for valuation rings of function fields.

INPUT:

- `field` – function field

- `place` – place of the function field

EXAMPLES:

```
sage: K.<x> = FunctionField(GF(2)); _.<Y> = K[]
sage: L.<y> = K.extension(Y^2 + Y + x + 1/x)
sage: p = L.places_finite()[0]
sage: p.valuation_ring()
Valuation ring at Place (x, x*y)
```

```
>>> from sage.all import *
>>> K = FunctionField(GF(Integer(2)), names=('x',)); (x,) = K._first_ngens(1); _ = K['Y']; (Y,) = _._first_ngens(1)
>>> L = K.extension(Y**Integer(2) + Y + x + Integer(1)/x, names=('y',)); (y,) = L._first_ngens(1)
>>> p = L.places_finite()[Integer(0)]
>>> p.valuation_ring()
Valuation ring at Place (x, x*y)
```

place()

Return the place associated with the valuation ring.

EXAMPLES:

```
sage: K.<x> = FunctionField(GF(2)); _.<Y> = K[]
sage: L.<y> = K.extension(Y^2 + Y + x + 1/x)
sage: p = L.places_finite()[0]
sage: R = p.valuation_ring()
sage: p == R.place()
True
```

```
>>> from sage.all import *
>>> K = FunctionField(GF(Integer(2)), names=('x',)); (x,) = K._first_ngens(1); _ = K['Y']; (Y,) = _._first_ngens(1)
>>> L = K.extension(Y**Integer(2) + Y + x + Integer(1)/x, names=('y',)); (y,) = L._first_ngens(1)
>>> p = L.places_finite()[Integer(0)]
>>> R = p.valuation_ring()
>>> p == R.place()
True
```

residue_field(name=None)

Return the residue field of the valuation ring together with the maps from and to it.

INPUT:

- `name` – string; name of the generator of the field

OUTPUT:

- a field isomorphic to the residue field
- a ring homomorphism from the valuation ring to the field
- a ring homomorphism from the field to the valuation ring

EXAMPLES:

```

sage: K.<x> = FunctionField(GF(2)); _.<Y> = K[]
sage: L.<y> = K.extension(Y^2 + Y + x + 1/x)
sage: p = L.places_finite()[0]
sage: R = p.valuation_ring()
sage: k, fr_k, to_k = R.residue_field()
sage: k
Finite Field of size 2
sage: fr_k
Ring morphism:
  From: Finite Field of size 2
  To:   Valuation ring at Place (x, x*y)
sage: to_k
Ring morphism:
  From: Valuation ring at Place (x, x*y)
  To:   Finite Field of size 2
sage: to_k(1/y)
0
sage: to_k(y/(1+y))
1

```

```

>>> from sage.all import *
>>> K = FunctionField(GF(Integer(2)), names=('x',)); (x,) = K._first_ngens(1);
↪ _ = K['Y']; (Y,) = _._first_ngens(1)
>>> L = K.extension(Y**Integer(2) + Y + x + Integer(1)/x, names=('y',)); (y,) =
↪ L._first_ngens(1)
>>> p = L.places_finite()[Integer(0)]
>>> R = p.valuation_ring()
>>> k, fr_k, to_k = R.residue_field()
>>> k
Finite Field of size 2
>>> fr_k
Ring morphism:
  From: Finite Field of size 2
  To:   Valuation ring at Place (x, x*y)
>>> to_k
Ring morphism:
  From: Valuation ring at Place (x, x*y)
  To:   Finite Field of size 2
>>> to_k(Integer(1)/y)
0
>>> to_k(y/(Integer(1)+y))
1

```

DERIVATIONS OF FUNCTION FIELDS

For global function fields, which have positive characteristics, the higher derivation is available:

```
sage: K.<x> = FunctionField(GF(2)); _.<Y> = K[]
sage: L.<y> = K.extension(Y^3 + x + x^3*Y) #_
↳needs sage.rings.function_field
sage: h = L.higher_derivation() #_
↳needs sage.rings.function_field
sage: h(y^2, 2) #_
↳needs sage.rings.function_field
((x^7 + 1)/x^2)*y^2 + x^3*y
```

```
>>> from sage.all import *
>>> K = FunctionField(GF(Integer(2)), names=('x',)); (x,) = K._first_ngens(1); _ = K[
↳'Y']; (Y,) = _._first_ngens(1)
>>> L = K.extension(Y**Integer(3) + x + x**Integer(3)*Y, names=('y',)); (y,) = L._
↳first_ngens(1) # needs sage.rings.function_field
>>> h = L.higher_derivation() #_
↳needs sage.rings.function_field
>>> h(y**Integer(2), Integer(2))
↳ # needs sage.rings.function_field
((x^7 + 1)/x^2)*y^2 + x^3*y
```

AUTHORS:

- William Stein (2010): initial version
- Julian R  th (2011-09-14, 2014-06-23, 2017-08-21): refactored class hierarchy; added derivation classes; morphisms to/from fraction fields
- Kwankyu Lee (2017-04-30): added higher derivations and completions

class sage.rings.function_field.derivations.**FunctionFieldDerivation** (*parent*)

Bases: RingDerivationWithoutTwist

Base class for derivations on function fields.

A derivation on R is a map $R \rightarrow R$ with $D(\alpha + \beta) = D(\alpha) + D(\beta)$ and $D(\alpha\beta) = \beta D(\alpha) + \alpha D(\beta)$ for all $\alpha, \beta \in R$.

EXAMPLES:

```
sage: K.<x> = FunctionField(QQ)
sage: d = K.derivation()
```

(continues on next page)

(continued from previous page)

```
sage: d
d/dx
```

```
>>> from sage.all import *
>>> K = FunctionField(QQ, names=('x',)); (x,) = K._first_ngens(1)
>>> d = K.derivation()
>>> d
d/dx
```

is_injective()

Return False since a derivation is never injective.

EXAMPLES:

```
sage: K.<x> = FunctionField(QQ)
sage: d = K.derivation()
sage: d.is_injective()
False
```

```
>>> from sage.all import *
>>> K = FunctionField(QQ, names=('x',)); (x,) = K._first_ngens(1)
>>> d = K.derivation()
>>> d.is_injective()
False
```


DERIVATIONS OF FUNCTION FIELDS: RATIONAL

```
class sage.rings.function_field.derivations_rational.FunctionFieldDerivation_rational (parent,  
                                                                                   u=None)
```

Bases: *FunctionFieldDerivation*

Derivations on rational function fields.

EXAMPLES:

```
sage: K.<x> = FunctionField(QQ)
sage: K.derivation()
d/dx
```

```
>>> from sage.all import *
>>> K = FunctionField(QQ, names=('x',)); (x,) = K._first_ngens(1)
>>> K.derivation()
d/dx
```


DERIVATIONS OF FUNCTION FIELDS: EXTENSION

```
class sage.rings.function_field.derivations_polymod.FunctionFieldDerivation_inseparable(parent,
                                                                                       u=None)
```

Bases: *FunctionFieldDerivation*

Initialize this derivation.

INPUT:

- parent – the parent of this derivation
- u – a parameter describing the derivation

EXAMPLES:

```
sage: K.<x> = FunctionField(GF(2))
sage: R.<y> = K[]
sage: L.<y> = K.extension(y^2 - x)
sage: d = L.derivation()
```

```
>>> from sage.all import *
>>> K = FunctionField(GF(Integer(2)), names=('x',)); (x,) = K._first_ngens(1)
>>> R = K['y']; (y,) = R._first_ngens(1)
>>> L = K.extension(y**Integer(2) - x, names=('y',)); (y,) = L._first_ngens(1)
>>> d = L.derivation()
```

This also works for iterated non-monic extensions:

```
sage: K.<x> = FunctionField(GF(2))
sage: R.<y> = K[]
sage: L.<y> = K.extension(y^2 - 1/x)
sage: R.<z> = L[]
sage: M.<z> = L.extension(z^2*y - x^3)
sage: M.derivation()
d/dz
```

```
>>> from sage.all import *
>>> K = FunctionField(GF(Integer(2)), names=('x',)); (x,) = K._first_ngens(1)
>>> R = K['y']; (y,) = R._first_ngens(1)
>>> L = K.extension(y**Integer(2) - Integer(1)/x, names=('y',)); (y,) = L._first_
↳ngens(1)
>>> R = L['z']; (z,) = R._first_ngens(1)
```

(continues on next page)

(continued from previous page)

```
>>> M = L.extension(z**Integer(2)*y - x**Integer(3), names=('z',)); (z,) = M._
↪first_ngens(1)
>>> M.derivation()
d/dz
```

We can also create a multiple of the canonical derivation:

```
sage: M.derivation([x])
x*d/dz
```

```
>>> from sage.all import *
>>> M.derivation([x])
x*d/dz
```

```
class sage.rings.function_field.derivations_polymod.FunctionFieldDerivation_separable (par-
ent,
d)
```

Bases: *FunctionFieldDerivation*

Derivations of separable extensions.

EXAMPLES:

```
sage: K.<x> = FunctionField(QQ)
sage: R.<y> = K[]
sage: L.<y> = K.extension(y^2 - x)
sage: L.derivation()
d/dx
```

```
>>> from sage.all import *
>>> K = FunctionField(QQ, names=('x',)); (x,) = K._first_ngens(1)
>>> R = K['y']; (y,) = R._first_ngens(1)
>>> L = K.extension(y**Integer(2) - x, names=('y',)); (y,) = L._first_ngens(1)
>>> L.derivation()
d/dx
```

```
class sage.rings.function_field.derivations_polymod.FunctionFieldHigherDerivation (field)
```

Bases: *Map*

Base class of higher derivations on function fields.

INPUT:

- *field* – function field on which the derivation operates

EXAMPLES:

```
sage: F.<x> = FunctionField(GF(2))
sage: F.higher_derivation()
Higher derivation map:
From: Rational function field in x over Finite Field of size 2
To: Rational function field in x over Finite Field of size 2
```

```

>>> from sage.all import *
>>> F = FunctionField(GF(Integer(2)), names=('x',)); (x,) = F._first_ngens(1)
>>> F.higher_derivation()
Higher derivation map:
  From: Rational function field in x over Finite Field of size 2
  To:   Rational function field in x over Finite Field of size 2

```

class sage.rings.function_field.derivations_polymod.**FunctionFieldHigherDerivation_char_zero** (*field*)

Bases: *FunctionFieldHigherDerivation*

Higher derivations of function fields of characteristic zero.

INPUT:

- *field* – function field on which the derivation operates

EXAMPLES:

```

sage: K.<x> = FunctionField(QQ); _.<Y> = K[]
sage: L.<y> = K.extension(Y^3 + x + x^3*Y)
sage: h = L.higher_derivation()
sage: h
Higher derivation map:
  From: Function field in y defined by y^3 + x^3*y + x
  To:   Function field in y defined by y^3 + x^3*y + x
sage: h(y,1) == -(3*x^2*y+1)/(3*y^2+x^3)
True
sage: h(y^2,1) == -2*y*(3*x^2*y+1)/(3*y^2+x^3)
True
sage: e = L.random_element()
sage: h(h(e,1),1) == 2*h(e,2)
True
sage: h(h(h(e,1),1),1) == 3*2*h(e,3)
True

```

```

>>> from sage.all import *
>>> K = FunctionField(QQ, names=('x',)); (x,) = K._first_ngens(1); _ = K['Y']; (Y,
↪) = _._first_ngens(1)
>>> L = K.extension(Y**Integer(3) + x + x**Integer(3)*Y, names=('y',)); (y,) = L._
↪first_ngens(1)
>>> h = L.higher_derivation()
>>> h
Higher derivation map:
  From: Function field in y defined by y^3 + x^3*y + x
  To:   Function field in y defined by y^3 + x^3*y + x
>>> h(y,Integer(1)) == -(Integer(3)*x**Integer(2)*y+Integer(1))/
↪(Integer(3)*y**Integer(2)+x**Integer(3))
True
>>> h(y**Integer(2),Integer(1)) == -
↪Integer(2)*y*(Integer(3)*x**Integer(2)*y+Integer(1))/
↪(Integer(3)*y**Integer(2)+x**Integer(3))
True
>>> e = L.random_element()
>>> h(h(e,Integer(1)),Integer(1)) == Integer(2)*h(e,Integer(2))

```

(continues on next page)

(continued from previous page)

```
True
>>> h(h(h(e, Integer(1)), Integer(1)), Integer(1)) == Integer(3)*Integer(2)*h(e,
↪ Integer(3))
True
```

class sage.rings.function_field.derivations_polymod.**FunctionFieldHigherDerivation_global** (*field*)
 Bases: *FunctionFieldHigherDerivation*

Higher derivations of global function fields.

INPUT:

- *field* – function field on which the derivation operates

EXAMPLES:

```
sage: K.<x> = FunctionField(GF(2)); _.<Y> = K[]
sage: L.<y> = K.extension(Y^3 + x + x^3*Y)
sage: h = L.higher_derivation()
sage: h
Higher derivation map:
  From: Function field in y defined by y^3 + x^3*y + x
  To:   Function field in y defined by y^3 + x^3*y + x
sage: h(y^2, 2)
((x^7 + 1)/x^2)*y^2 + x^3*y
```

```
>>> from sage.all import *
>>> K = FunctionField(GF(Integer(2)), names=('x',)); (x,) = K._first_ngens(1); _↪
↪ = K['Y']; (Y,) = _._first_ngens(1)
>>> L = K.extension(Y**Integer(3) + x + x**Integer(3)*Y, names=('y',)); (y,) = L._↪
↪ first_ngens(1)
>>> h = L.higher_derivation()
>>> h
Higher derivation map:
  From: Function field in y defined by y^3 + x^3*y + x
  To:   Function field in y defined by y^3 + x^3*y + x
>>> h(y**Integer(2), Integer(2))
((x^7 + 1)/x^2)*y^2 + x^3*y
```

class sage.rings.function_field.derivations_polymod.**RationalFunctionFieldHigherDerivation_global** (*field*)
 Bases: *FunctionFieldHigherDerivation*

Higher derivations of rational function fields over finite fields.

INPUT:

- *field* – function field on which the derivation operates

EXAMPLES:

```
sage: F.<x> = FunctionField(GF(2))
sage: h = F.higher_derivation()
sage: h
Higher derivation map:
  From: Rational function field in x over Finite Field of size 2
  To:   Rational function field in x over Finite Field of size 2
```

(continues on next page)

(continued from previous page)

```
sage: h(x^2, 2)
```

```
1
```

```
>>> from sage.all import *
>>> F = FunctionField(GF(Integer(2)), names=('x',)); (x,) = F._first_ngens(1)
>>> h = F.higher_derivation()
>>> h
Higher derivation map:
  From: Rational function field in x over Finite Field of size 2
  To:   Rational function field in x over Finite Field of size 2
>>> h(x**Integer(2), Integer(2))
```

```
1
```


MORPHISMS OF FUNCTION FIELDS

Maps and morphisms useful for computations with function fields.

EXAMPLES:

```
sage: K.<x> = FunctionField(QQ); R.<y> = K[]
sage: K.hom(1/x)
Function Field endomorphism of Rational function field in x over Rational Field
Defn: x |--> 1/x

sage: # needs sage.rings.function_field
sage: L.<y> = K.extension(y^2 - x)
sage: K.hom(y)
Function Field morphism:
  From: Rational function field in x over Rational Field
  To:   Function field in y defined by y^2 - x
  Defn: x |--> y
sage: L.hom([y,x])
Function Field endomorphism of Function field in y defined by y^2 - x
Defn: y |--> y
      x |--> x
sage: L.hom([x,y])
Traceback (most recent call last):
...
ValueError: invalid morphism
```

```
>>> from sage.all import *
>>> K = FunctionField(QQ, names=('x',)); (x,) = K._first_ngens(1); R = K['y']; (y,) = R._first_ngens(1)
>>> K.hom(Integer(1)/x)
Function Field endomorphism of Rational function field in x over Rational Field
Defn: x |--> 1/x

>>> # needs sage.rings.function_field
>>> L = K.extension(y**Integer(2) - x, names=('y',)); (y,) = L._first_ngens(1)
>>> K.hom(y)
Function Field morphism:
  From: Rational function field in x over Rational Field
  To:   Function field in y defined by y^2 - x
  Defn: x |--> y
>>> L.hom([y,x])
Function Field endomorphism of Function field in y defined by y^2 - x
```

(continues on next page)

(continued from previous page)

```

Defn: y |--> y
      x |--> x
>>> L.hom([x,y])
Traceback (most recent call last):
...
ValueError: invalid morphism

```

AUTHORS:

- William Stein (2010): initial version
- Julian R  th (2011-09-14, 2014-06-23, 2017-08-21): refactored class hierarchy; added derivation classes; morphisms to/from fraction fields
- Kwankyu Lee (2017-04-30): added higher derivations and completions

class sage.rings.function_field.maps.**FractionFieldToFunctionField**

Bases: *FunctionFieldVectorSpaceIsomorphism*

Isomorphism from a fraction field of a polynomial ring to the isomorphic function field.

EXAMPLES:

```

sage: K = QQ['x'].fraction_field()
sage: L = K.function_field()
sage: f = L.coerce_map_from(K); f
Isomorphism:
  From: Fraction Field of Univariate Polynomial Ring in x over Rational Field
  To:   Rational function field in x over Rational Field

```

```

>>> from sage.all import *
>>> K = QQ['x'].fraction_field()
>>> L = K.function_field()
>>> f = L.coerce_map_from(K); f
Isomorphism:
  From: Fraction Field of Univariate Polynomial Ring in x over Rational Field
  To:   Rational function field in x over Rational Field

```

See also

FunctionFieldToFractionField

section()

Return the inverse map of this isomorphism.

EXAMPLES:

```

sage: K = QQ['x'].fraction_field()
sage: L = K.function_field()
sage: f = L.coerce_map_from(K)
sage: f.section()
Isomorphism:
  From: Rational function field in x over Rational Field

```

(continues on next page)

(continued from previous page)

```
To: Fraction Field of Univariate Polynomial Ring in x over Rational
↪Field
```

```
>>> from sage.all import *
>>> K = QQ['x'].fraction_field()
>>> L = K.function_field()
>>> f = L.coerce_map_from(K)
>>> f.section()
Isomorphism:
  From: Rational function field in x over Rational Field
  To: Fraction Field of Univariate Polynomial Ring in x over Rational
↪Field
```

```
class sage.rings.function_field.maps.FunctionFieldCompletion (field, place, name=None,
                                                             prec=None, gen_name=None)
```

Bases: Map

Completions on function fields.

INPUT:

- field – function field
- place – place of the function field
- name – string for the name of the series variable
- prec – positive integer; default precision
- gen_name – string; name of the generator of the residue field; used only when place is non-rational

EXAMPLES:

```
sage: # needs sage.rings.finite_rings sage.rings.function_field
sage: K.<x> = FunctionField(GF(2)); _.<Y> = K[]
sage: L.<y> = K.extension(Y^2 + Y + x + 1/x)
sage: p = L.places_finite()[0]
sage: m = L.completion(p)
sage: m
Completion map:
  From: Function field in y defined by y^2 + y + (x^2 + 1)/x
  To: Laurent Series Ring in s over Finite Field of size 2
sage: m(x)
s^2 + s^3 + s^4 + s^5 + s^7 + s^8 + s^9 + s^10 + s^12 + s^13
+ s^15 + s^16 + s^17 + s^19 + O(s^22)
sage: m(y)
s^-1 + 1 + s^3 + s^5 + s^7 + s^9 + s^13 + s^15 + s^17 + O(s^19)
sage: m(x*y) == m(x) * m(y)
True
sage: m(x+y) == m(x) + m(y)
True
```

```
>>> from sage.all import *
>>> # needs sage.rings.finite_rings sage.rings.function_field
>>> K = FunctionField(GF(Integer(2)), names=('x',)); (x,) = K._first_ngens(1); _
```

(continues on next page)

(continued from previous page)

```

↪= K['Y']; (Y,) = __.first_ngens(1)
>>> L = K.extension(Y**Integer(2) + Y + x + Integer(1)/x, names=('y',)); (y,) = L.
↪_first_ngens(1)
>>> p = L.places_finite()[Integer(0)]
>>> m = L.completion(p)
>>> m
Completion map:
  From: Function field in y defined by y^2 + y + (x^2 + 1)/x
  To:   Laurent Series Ring in s over Finite Field of size 2
>>> m(x)
s^2 + s^3 + s^4 + s^5 + s^7 + s^8 + s^9 + s^10 + s^12 + s^13
+ s^15 + s^16 + s^17 + s^19 + O(s^22)
>>> m(y)
s^-1 + 1 + s^3 + s^5 + s^7 + s^9 + s^13 + s^15 + s^17 + O(s^19)
>>> m(x*y) == m(x) * m(y)
True
>>> m(x+y) == m(x) + m(y)
True

```

The variable name of the series can be supplied. If the place is not rational such that the residue field is a proper extension of the constant field, you can also specify the generator name of the extension:

```

sage: # needs sage.rings.finite_rings sage.rings.function_field
sage: p2 = L.places_finite(2)[0]
sage: p2
Place (x^2 + x + 1, x*y + 1)
sage: m2 = L.completion(p2, 't', gen_name='b')
sage: m2(x)
(b + 1) + t + t^2 + t^4 + t^8 + t^16 + O(t^20)
sage: m2(y)
b + b*t + b*t^3 + b*t^4 + (b + 1)*t^5 + (b + 1)*t^7 + b*t^9 + b*t^11
+ b*t^12 + b*t^13 + b*t^15 + b*t^16 + (b + 1)*t^17 + (b + 1)*t^19 + O(t^20)

```

```

>>> from sage.all import *
>>> # needs sage.rings.finite_rings sage.rings.function_field
>>> p2 = L.places_finite(Integer(2))[Integer(0)]
>>> p2
Place (x^2 + x + 1, x*y + 1)
>>> m2 = L.completion(p2, 't', gen_name='b')
>>> m2(x)
(b + 1) + t + t^2 + t^4 + t^8 + t^16 + O(t^20)
>>> m2(y)
b + b*t + b*t^3 + b*t^4 + (b + 1)*t^5 + (b + 1)*t^7 + b*t^9 + b*t^11
+ b*t^12 + b*t^13 + b*t^15 + b*t^16 + (b + 1)*t^17 + (b + 1)*t^19 + O(t^20)

```

default_precision()

Return the default precision.

EXAMPLES:

```

sage: # needs sage.rings.finite_rings sage.rings.function_field
sage: K.<x> = FunctionField(GF(2)); __.<Y> = K[]

```

(continues on next page)

(continued from previous page)

```

sage: L.<y> = K.extension(Y^2 + Y + x + 1/x)
sage: p = L.places_finite()[0]
sage: m = L.completion(p)
sage: m.default_precision()
20

```

```

>>> from sage.all import *
>>> # needs sage.rings.finite_rings sage.rings.function_field
>>> K = FunctionField(GF(Integer(2)), names=('x',)); (x,) = K._first_ngens(1);
↪ _ = K['Y']; (Y,) = _._first_ngens(1)
>>> L = K.extension(Y**Integer(2) + Y + x + Integer(1)/x, names=('y',)); (y,) ↪
↪ = L._first_ngens(1)
>>> p = L.places_finite()[Integer(0)]
>>> m = L.completion(p)
>>> m.default_precision()
20

```

class sage.rings.function_field.maps.**FunctionFieldConversionToConstantBaseField**(parent)

Bases: Map

Conversion map from the function field to its constant base field.

EXAMPLES:

```

sage: K.<x> = FunctionField(QQ)
sage: QQ.convert_map_from(K)
Conversion map:
From: Rational function field in x over Rational Field
To: Rational Field

```

```

>>> from sage.all import *
>>> K = FunctionField(QQ, names=('x',)); (x,) = K._first_ngens(1)
>>> QQ.convert_map_from(K)
Conversion map:
From: Rational function field in x over Rational Field
To: Rational Field

```

class sage.rings.function_field.maps.**FunctionFieldLinearMap**

Bases: SetMorphism

Linear map to function fields.

class sage.rings.function_field.maps.**FunctionFieldLinearMapSection**

Bases: SetMorphism

Section of linear map from function fields.

class sage.rings.function_field.maps.**FunctionFieldMorphism**(parent, im_gen, base_morphism)

Bases: RingHomomorphism

Base class for morphisms between function fields.

EXAMPLES:

```
sage: K.<x> = FunctionField(QQ)
sage: f = K.hom(1/x); f
Function Field endomorphism of Rational function field in x over Rational Field
Defn: x |--> 1/x
```

```
>>> from sage.all import *
>>> K = FunctionField(QQ, names=('x',)); (x,) = K._first_ngens(1)
>>> f = K.hom(Integer(1)/x); f
Function Field endomorphism of Rational function field in x over Rational Field
Defn: x |--> 1/x
```

```
class sage.rings.function_field.maps.FunctionFieldMorphism_polymod(parent, im_gen,
                                                                    base_morphism)
```

Bases: *FunctionFieldMorphism*

Morphism from a finite extension of a function field to a function field.

EXAMPLES:

```
sage: # needs sage.rings.finite_rings sage.rings.function_field
sage: K.<x> = FunctionField(GF(7)); R.<y> = K[]
sage: L.<y> = K.extension(y^3 + 6*x^3 + x)
sage: f = L.hom(y^2); f
Function Field endomorphism of Function field in y defined by y^3 + 6*x^3 + x
Defn: y |--> 2*y
sage: factor(L.polynomial())
y^3 + 6*x^3 + x
sage: f(y).charpoly('y')
y^3 + 6*x^3 + x
```

```
>>> from sage.all import *
>>> # needs sage.rings.finite_rings sage.rings.function_field
>>> K = FunctionField(GF(Integer(7)), names=('x',)); (x,) = K._first_ngens(1); R.
↳ K['y']; (y,) = R._first_ngens(1)
>>> L = K.extension(y**Integer(3) + Integer(6)*x**Integer(3) + x, names=('y',));
↳ (y,) = L._first_ngens(1)
>>> f = L.hom(y*Integer(2)); f
Function Field endomorphism of Function field in y defined by y^3 + 6*x^3 + x
Defn: y |--> 2*y
>>> factor(L.polynomial())
y^3 + 6*x^3 + x
>>> f(y).charpoly('y')
y^3 + 6*x^3 + x
```

```
class sage.rings.function_field.maps.FunctionFieldMorphism_rational(parent, im_gen,
                                                                    base_morphism)
```

Bases: *FunctionFieldMorphism*

Morphism from a rational function field to a function field.

```
class sage.rings.function_field.maps.FunctionFieldRingMorphism
```

Bases: *SetMorphism*

Ring homomorphism.

class sage.rings.function_field.maps.**FunctionFieldToFractionField**

Bases: *FunctionFieldVectorSpaceIsomorphism*

Isomorphism from rational function field to the isomorphic fraction field of a polynomial ring.

EXAMPLES:

```
sage: K = QQ['x'].fraction_field()
sage: L = K.function_field()
sage: f = K.coerce_map_from(L); f
Isomorphism:
  From: Rational function field in x over Rational Field
  To:   Fraction Field of Univariate Polynomial Ring in x over Rational Field
```

```
>>> from sage.all import *
>>> K = QQ['x'].fraction_field()
>>> L = K.function_field()
>>> f = K.coerce_map_from(L); f
Isomorphism:
  From: Rational function field in x over Rational Field
  To:   Fraction Field of Univariate Polynomial Ring in x over Rational Field
```

See also

FractionFieldToFunctionField

section()

Return the inverse map of this isomorphism.

EXAMPLES:

```
sage: K = QQ['x'].fraction_field()
sage: L = K.function_field()
sage: f = K.coerce_map_from(L)
sage: f.section()
Isomorphism:
  From: Fraction Field of Univariate Polynomial Ring in x over Rational_
↪Field
  To:   Rational function field in x over Rational Field
```

```
>>> from sage.all import *
>>> K = QQ['x'].fraction_field()
>>> L = K.function_field()
>>> f = K.coerce_map_from(L)
>>> f.section()
Isomorphism:
  From: Fraction Field of Univariate Polynomial Ring in x over Rational_
↪Field
  To:   Rational function field in x over Rational Field
```

class sage.rings.function_field.maps.**FunctionFieldVectorSpaceIsomorphism**

Bases: *Morphism*

Base class for isomorphisms between function fields and vector spaces.

EXAMPLES:

```
sage: # needs sage.rings.function_field
sage: K.<x> = FunctionField(QQ); R.<y> = K[]
sage: L.<y> = K.extension(y^2 - x*y + 4*x^3)
sage: V, f, t = L.vector_space()
sage: isinstance(f, sage.rings.function_field.maps.
↪FunctionFieldVectorSpaceIsomorphism)
True
```

```
>>> from sage.all import *
>>> # needs sage.rings.function_field
>>> K = FunctionField(QQ, names=('x',)); (x,) = K._first_ngens(1); R = K['y']; (y,
↪) = R._first_ngens(1)
>>> L = K.extension(y**Integer(2) - x*y + Integer(4)*x**Integer(3), names=('y',));
↪ (y,) = L._first_ngens(1)
>>> V, f, t = L.vector_space()
>>> isinstance(f, sage.rings.function_field.maps.
↪FunctionFieldVectorSpaceIsomorphism)
True
```

is_injective()

Return True, since the isomorphism is injective.

EXAMPLES:

```
sage: # needs sage.rings.function_field
sage: K.<x> = FunctionField(QQ); R.<y> = K[]
sage: L.<y> = K.extension(y^2 - x*y + 4*x^3)
sage: V, f, t = L.vector_space()
sage: f.is_injective()
True
```

```
>>> from sage.all import *
>>> # needs sage.rings.function_field
>>> K = FunctionField(QQ, names=('x',)); (x,) = K._first_ngens(1); R = K['y'];
↪ (y,) = R._first_ngens(1)
>>> L = K.extension(y**Integer(2) - x*y + Integer(4)*x**Integer(3), names=('y
↪',)); (y,) = L._first_ngens(1)
>>> V, f, t = L.vector_space()
>>> f.is_injective()
True
```

is_surjective()

Return True, since the isomorphism is surjective.

EXAMPLES:

```
sage: # needs sage.rings.function_field
sage: K.<x> = FunctionField(QQ); R.<y> = K[]
sage: L.<y> = K.extension(y^2 - x*y + 4*x^3)
sage: V, f, t = L.vector_space()
sage: f.is_surjective()
True
```



```

>>> from sage.all import *
>>> # needs sage.rings.function_field
>>> K = FunctionField(QQ, names=('x',)); (x,) = K._first_ngens(1); R = K['y'];
↪ (y,) = R._first_ngens(1)
>>> L = K.extension(y**Integer(2) - x*y + Integer(4)*x**Integer(3), names=('y',)); (y,) = L._first_ngens(1)
>>> V, f, t = L.vector_space()
>>> f.is_surjective()
True

```

class sage.rings.function_field.maps.**MapFunctionFieldToVectorSpace**(K, V)

Bases: *FunctionFieldVectorSpaceIsomorphism*

Isomorphism from a function field to a vector space.

EXAMPLES:

```

sage: # needs sage.rings.function_field
sage: K.<x> = FunctionField(QQ); R.<y> = K[]
sage: L.<y> = K.extension(y^2 - x*y + 4*x^3)
sage: V, f, t = L.vector_space(); t
Isomorphism:
  From: Function field in y defined by y^2 - x*y + 4*x^3
  To:   Vector space of dimension 2 over Rational function field in x over_
↪Rational Field

```

```

>>> from sage.all import *
>>> # needs sage.rings.function_field
>>> K = FunctionField(QQ, names=('x',)); (x,) = K._first_ngens(1); R = K['y']; (y,
↪) = R._first_ngens(1)
>>> L = K.extension(y**Integer(2) - x*y + Integer(4)*x**Integer(3), names=('y',));
↪ (y,) = L._first_ngens(1)
>>> V, f, t = L.vector_space(); t
Isomorphism:
  From: Function field in y defined by y^2 - x*y + 4*x^3
  To:   Vector space of dimension 2 over Rational function field in x over_
↪Rational Field

```

class sage.rings.function_field.maps.**MapVectorSpaceToFunctionField**(V, K)

Bases: *FunctionFieldVectorSpaceIsomorphism*

Isomorphism from a vector space to a function field.

EXAMPLES:

```

sage: # needs sage.rings.function_field
sage: K.<x> = FunctionField(QQ); R.<y> = K[]
sage: L.<y> = K.extension(y^2 - x*y + 4*x^3)
sage: V, f, t = L.vector_space(); f
Isomorphism:
  From: Vector space of dimension 2 over Rational function field in x over_
↪Rational Field
  To:   Function field in y defined by y^2 - x*y + 4*x^3

```

```

>>> from sage.all import *
>>> # needs sage.rings.function_field
>>> K = FunctionField(QQ, names=('x',)); (x,) = K._first_ngens(1); R = K['y']; (y,
↪) = R._first_ngens(1)
>>> L = K.extension(y**Integer(2) - x*y + Integer(4)*x**Integer(3), names=('y',));
↪ (y,) = L._first_ngens(1)
>>> V, f, t = L.vector_space(); f
Isomorphism:
  From: Vector space of dimension 2 over Rational function field in x over
↪ Rational Field
  To:   Function field in y defined by y^2 - x*y + 4*x^3

```

codomain()

Return the function field which is the codomain of the isomorphism.

EXAMPLES:

```

sage: # needs sage.rings.function_field
sage: K.<x> = FunctionField(QQ); R.<y> = K[]
sage: L.<y> = K.extension(y^2 - x*y + 4*x^3)
sage: V, f, t = L.vector_space()
sage: f.codomain()
Function field in y defined by y^2 - x*y + 4*x^3

```

```

>>> from sage.all import *
>>> # needs sage.rings.function_field
>>> K = FunctionField(QQ, names=('x',)); (x,) = K._first_ngens(1); R = K['y'];
↪ (y,) = R._first_ngens(1)
>>> L = K.extension(y**Integer(2) - x*y + Integer(4)*x**Integer(3), names=('y
↪ ',)); (y,) = L._first_ngens(1)
>>> V, f, t = L.vector_space()
>>> f.codomain()
Function field in y defined by y^2 - x*y + 4*x^3

```

domain()

Return the vector space which is the domain of the isomorphism.

EXAMPLES:

```

sage: # needs sage.rings.function_field
sage: K.<x> = FunctionField(QQ); R.<y> = K[]
sage: L.<y> = K.extension(y^2 - x*y + 4*x^3)
sage: V, f, t = L.vector_space()
sage: f.domain()
Vector space of dimension 2 over Rational function field in x over Rational
↪ Field

```

```

>>> from sage.all import *
>>> # needs sage.rings.function_field
>>> K = FunctionField(QQ, names=('x',)); (x,) = K._first_ngens(1); R = K['y'];
↪ (y,) = R._first_ngens(1)
>>> L = K.extension(y**Integer(2) - x*y + Integer(4)*x**Integer(3), names=('y
↪ ',)); (y,) = L._first_ngens(1)

```

(continues on next page)

(continued from previous page)

```
>>> V, f, t = L.vector_space()
>>> f.domain()
Vector space of dimension 2 over Rational function field in x over Rational
↳Field
```


SPECIAL EXTENSIONS OF FUNCTION FIELDS

This module currently implements only constant field extension.

24.1 Constant field extensions

EXAMPLES:

Constant field extension of the rational function field over rational numbers:

```
sage: K.<x> = FunctionField(QQ)
sage: N.<a> = QuadraticField(2)                                     #_
↪needs sage.rings.number_field
sage: L = K.extension_constant_field(N)                           #_
↪needs sage.rings.number_field
sage: L                                                         #_
↪needs sage.rings.number_field
Rational function field in x over Number Field in a with defining
polynomial x^2 - 2 with a = 1.4142... over its base
sage: d = (x^2 - 2).divisor()                                     #_
↪needs sage.libs.pari sage.modules
sage: d                                                         #_
↪needs sage.libs.pari sage.modules
-2*Place (1/x)
+ Place (x^2 - 2)
sage: L.conorm_divisor(d)                                         #_
↪needs sage.libs.pari sage.modules sage.rings.number_field
-2*Place (1/x)
+ Place (x - a)
+ Place (x + a)
```

```
>>> from sage.all import *
>>> K = FunctionField(QQ, names=('x',)); (x,) = K._first_ngens(1)
>>> N = QuadraticField(Integer(2), names=('a',)); (a,) = N._first_ngens(1) # needs_
↪sage.rings.number_field
>>> L = K.extension_constant_field(N)                             #_
↪needs sage.rings.number_field
>>> L                                                         #_
↪needs sage.rings.number_field
Rational function field in x over Number Field in a with defining
polynomial x^2 - 2 with a = 1.4142... over its base
>>> d = (x**Integer(2) - Integer(2)).divisor()                   _
```

(continues on next page)

(continued from previous page)

```

↪ # needs sage.libs.pari sage.modules
>>> d #_
↪ needs sage.libs.pari sage.modules
-2*Place (1/x)
+ Place (x^2 - 2)
>>> L.conorm_divisor(d) #_
↪ needs sage.libs.pari sage.modules sage.rings.number_field
-2*Place (1/x)
+ Place (x - a)
+ Place (x + a)

```

Constant field extension of a function field over a finite field:

```

sage: # needs sage.rings.finite_rings sage.rings.function_field
sage: K.<x> = FunctionField(GF(2)); R.<Y> = K[]
sage: F.<y> = K.extension(Y^3 - x^2*(x^2 + x + 1)^2)
sage: E = F.extension_constant_field(GF(2^3))
sage: E
Function field in y defined by y^3 + x^6 + x^4 + x^2 over its base
sage: p = F.get_place(3)
sage: E.conorm_place(p) # random
Place (x + z3, y + z3^2 + z3)
+ Place (x + z3^2, y + z3)
+ Place (x + z3^2 + z3, y + z3^2)
sage: q = F.get_place(2)
sage: E.conorm_place(q) # random
Place (x + 1, y^2 + y + 1)
sage: E.conorm_divisor(p + q) # random
Place (x + 1, y^2 + y + 1)
+ Place (x + z3, y + z3^2 + z3)
+ Place (x + z3^2, y + z3)
+ Place (x + z3^2 + z3, y + z3^2)

```

```

>>> from sage.all import *
>>> # needs sage.rings.finite_rings sage.rings.function_field
>>> K = FunctionField(GF(Integer(2)), names=('x',)); (x,) = K._first_ngens(1); R = K[
↪ 'Y']; (Y,) = R._first_ngens(1)
>>> F = K.extension(Y**Integer(3) - x**Integer(2)*(x**Integer(2) + x +
↪ Integer(1))**Integer(2), names=('y',)); (y,) = F._first_ngens(1)
>>> E = F.extension_constant_field(GF(Integer(2)**Integer(3)))
>>> E
Function field in y defined by y^3 + x^6 + x^4 + x^2 over its base
>>> p = F.get_place(Integer(3))
>>> E.conorm_place(p) # random
Place (x + z3, y + z3^2 + z3)
+ Place (x + z3^2, y + z3)
+ Place (x + z3^2 + z3, y + z3^2)
>>> q = F.get_place(Integer(2))
>>> E.conorm_place(q) # random
Place (x + 1, y^2 + y + 1)
>>> E.conorm_divisor(p + q) # random
Place (x + 1, y^2 + y + 1)

```

(continues on next page)

(continued from previous page)

```

+ Place (x + z3, y + z3^2 + z3)
+ Place (x + z3^2, y + z3)
+ Place (x + z3^2 + z3, y + z3^2)

```

AUTHORS:

- Kwankyu Lee (2021-12-24): added constant field extension

class sage.rings.function_field.extensions.**ConstantFieldExtension** (F, k_ext)

Bases: *FunctionFieldExtension*

Constant field extension.

INPUT:

- F – a function field whose constant field is k
- k_ext – an extension of k

conorm_divisor (d)

Return the conorm of the divisor d in this extension.

INPUT:

- d – divisor of the base function field

OUTPUT: a divisor of the top function field

EXAMPLES:

```

sage: # needs sage.rings.finite_rings sage.rings.function_field
sage: K.<x> = FunctionField(GF(2)); R.<Y> = K[]
sage: F.<y> = K.extension(Y^3 - x^2*(x^2 + x + 1)^2)
sage: E = F.extension_constant_field(GF(2^3))
sage: p1 = F.get_place(3)
sage: p2 = F.get_place(2)
sage: c = E.conorm_divisor(2*p1 + 3*p2)
sage: c1 = E.conorm_place(p1)
sage: c2 = E.conorm_place(p2)
sage: c == 2*c1 + 3*c2
True

```

```

>>> from sage.all import *
>>> # needs sage.rings.finite_rings sage.rings.function_field
>>> K = FunctionField(GF(Integer(2)), names=('x',)); (x,) = K._first_ngens(1);
↪ R = K['Y']; (Y,) = R._first_ngens(1)
>>> F = K.extension(Y**Integer(3) - x**Integer(2)*(x**Integer(2) + x +
↪ Integer(1))**Integer(2), names=('y',)); (y,) = F._first_ngens(1)
>>> E = F.extension_constant_field(GF(Integer(2)**Integer(3)))
>>> p1 = F.get_place(Integer(3))
>>> p2 = F.get_place(Integer(2))
>>> c = E.conorm_divisor(Integer(2)*p1 + Integer(3)*p2)
>>> c1 = E.conorm_place(p1)
>>> c2 = E.conorm_place(p2)
>>> c == Integer(2)*c1 + Integer(3)*c2
True

```

conorm_place(*p*)

Return the conorm of the place *p* in this extension.

INPUT:

- *p* – place of the base function field

OUTPUT: divisor of the top function field

EXAMPLES:

```
sage: # needs sage.rings.finite_rings sage.rings.function_field
sage: K.<x> = FunctionField(GF(2)); R.<Y> = K[]
sage: F.<y> = K.extension(Y^3 - x^2*(x^2 + x + 1)^2)
sage: E = F.extension_constant_field(GF(2^3))
sage: p = F.get_place(3)
sage: d = E.conorm_place(p)
sage: [pl.degree() for pl in d.support()]
[1, 1, 1]
sage: p = F.get_place(2)
sage: d = E.conorm_place(p)
sage: [pl.degree() for pl in d.support()]
[2]
```

```
>>> from sage.all import *
>>> # needs sage.rings.finite_rings sage.rings.function_field
>>> K = FunctionField(GF(Integer(2)), names=('x',)); (x,) = K._first_ngens(1);
↳ R = K['Y']; (Y,) = R._first_ngens(1)
>>> F = K.extension(Y**Integer(3) - x**Integer(2)*(x**Integer(2) + x +
↳ Integer(1))**Integer(2), names=('y',)); (y,) = F._first_ngens(1)
>>> E = F.extension_constant_field(GF(Integer(2)**Integer(3)))
>>> p = F.get_place(Integer(3))
>>> d = E.conorm_place(p)
>>> [pl.degree() for pl in d.support()]
[1, 1, 1]
>>> p = F.get_place(Integer(2))
>>> d = E.conorm_place(p)
>>> [pl.degree() for pl in d.support()]
[2]
```

defining_morphism()

Return the defining morphism of this extension.

This is the morphism from the base to the top.

EXAMPLES:

```
sage: # needs sage.rings.finite_rings sage.rings.function_field
sage: K.<x> = FunctionField(GF(2)); R.<Y> = K[]
sage: F.<y> = K.extension(Y^3 - x^2*(x^2 + x + 1)^2)
sage: E = F.extension_constant_field(GF(2^3))
sage: E.defining_morphism()
Function Field morphism:
  From: Function field in y defined by y^3 + x^6 + x^4 + x^2
  To:   Function field in y defined by y^3 + x^6 + x^4 + x^2
  Defn: y |--> y
```

(continues on next page)

(continued from previous page)

```
x |--> x
1 |--> 1
```

```
>>> from sage.all import *
>>> # needs sage.rings.finite_rings sage.rings.function_field
>>> K = FunctionField(GF(Integer(2)), names=('x',)); (x,) = K._first_ngens(1);
↪ R = K['Y']; (Y,) = R._first_ngens(1)
>>> F = K.extension(Y**Integer(3) - x**Integer(2)*(x**Integer(2) + x +
↪ Integer(1))**Integer(2), names=('y',)); (y,) = F._first_ngens(1)
>>> E = F.extension_constant_field(GF(Integer(2)**Integer(3)))
>>> E.defined_morphism()
Function Field morphism:
  From: Function field in y defined by y^3 + x^6 + x^4 + x^2
  To:   Function field in y defined by y^3 + x^6 + x^4 + x^2
  Defn: y |--> y
        x |--> x
        1 |--> 1
```

top()

Return the top function field of this extension.

EXAMPLES:

```
sage: # needs sage.rings.finite_rings sage.rings.function_field
sage: K.<x> = FunctionField(GF(2)); R.<Y> = K[]
sage: F.<y> = K.extension(Y^3 - x^2*(x^2 + x + 1)^2)
sage: E = F.extension_constant_field(GF(2^3))
sage: E.top()
Function field in y defined by y^3 + x^6 + x^4 + x^2
```

```
>>> from sage.all import *
>>> # needs sage.rings.finite_rings sage.rings.function_field
>>> K = FunctionField(GF(Integer(2)), names=('x',)); (x,) = K._first_ngens(1);
↪ R = K['Y']; (Y,) = R._first_ngens(1)
>>> F = K.extension(Y**Integer(3) - x**Integer(2)*(x**Integer(2) + x +
↪ Integer(1))**Integer(2), names=('y',)); (y,) = F._first_ngens(1)
>>> E = F.extension_constant_field(GF(Integer(2)**Integer(3)))
>>> E.top()
Function field in y defined by y^3 + x^6 + x^4 + x^2
```

class sage.rings.function_field.extensions.**FunctionFieldExtension**Bases: `RingExtension_generic`

Abstract base class of function field extensions.

FACTORIES TO CONSTRUCT FUNCTION FIELDS

This module provides factories to construct function fields. These factories are only for internal use.

EXAMPLES:

```
sage: K.<x> = FunctionField(QQ); K
Rational function field in x over Rational Field
sage: L.<x> = FunctionField(QQ); L
Rational function field in x over Rational Field
sage: K is L
True
```

```
>>> from sage.all import *
>>> K = FunctionField(QQ, names=('x',)); (x,) = K._first_ngens(1); K
Rational function field in x over Rational Field
>>> L = FunctionField(QQ, names=('x',)); (x,) = L._first_ngens(1); L
Rational function field in x over Rational Field
>>> K is L
True
```

AUTHORS:

- William Stein (2010): initial version
- Maarten Derickx (2011-09-11): added `FunctionField_polymod_Constructor`, use `@cached_function`
- Julian Rueth (2011-09-14): replaced `@cached_function` with `UniqueFactory`

```
class sage.rings.function_field.constructor.FunctionFieldExtensionFactory
```

Bases: `UniqueFactory`

Create a function field defined as an extension of another function field by adjoining a root of a univariate polynomial. The returned function field is unique in the sense that if you call this function twice with an equal `polynomial` and `names` it returns the same python object in both calls.

INPUT:

- `polynomial` – univariate polynomial over a function field
- `names` – variable names (as a tuple of length 1 or string)
- `category` – category (defaults to category of function fields)

EXAMPLES:

```

sage: K.<x> = FunctionField(QQ)
sage: R.<y> = K[]
sage: y2 = y*1
sage: y2 is y
False
sage: L.<w> = K.extension(x - y^2) #_
↳needs sage.rings.function_field
sage: M.<w> = K.extension(x - y2^2) #_
↳needs sage.rings.function_field
sage: L is M #_
↳needs sage.rings.function_field
True

```

```

>>> from sage.all import *
>>> K = FunctionField(QQ, names=('x',)); (x,) = K._first_ngens(1)
>>> R = K['y']; (y,) = R._first_ngens(1)
>>> y2 = y*Integer(1)
>>> y2 is y
False
>>> L = K.extension(x - y**Integer(2), names=('w',)); (w,) = L._first_ngens(1) #_
↳needs sage.rings.function_field
>>> M = K.extension(x - y2**Integer(2), names=('w',)); (w,) = M._first_ngens(1) #_
↳needs sage.rings.function_field
>>> L is M #_
↳needs sage.rings.function_field
True

```

create_key (*polynomial*, *names*)

Given the arguments and keywords, create a key that uniquely determines this object.

EXAMPLES:

```

sage: K.<x> = FunctionField(QQ)
sage: R.<y> = K[]
sage: L.<w> = K.extension(x - y^2) # indirect doctest #_
↳needs sage.rings.function_field

```

```

>>> from sage.all import *
>>> K = FunctionField(QQ, names=('x',)); (x,) = K._first_ngens(1)
>>> R = K['y']; (y,) = R._first_ngens(1)
>>> L = K.extension(x - y**Integer(2), names=('w',)); (w,) = L._first_ngens(1)
↳# indirect doctest # needs sage.rings.function_field

```

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

Create the object from the key and extra arguments. This is only called if the object was not found in the cache.

EXAMPLES:

```

sage: K.<x> = FunctionField(QQ)
sage: R.<y> = K[]
sage: L.<w> = K.extension(x - y^2) # indirect doctest #_
↳needs sage.rings.function_field

```

(continues on next page)

(continued from previous page)

```

sage: y2 = y^1
sage: M.<w> = K.extension(x - y2^2) # indirect doctest #_
↳needs sage.rings.function_field
sage: L is M #_
↳needs sage.rings.function_field
True

```

```

>>> from sage.all import *
>>> K = FunctionField(QQ, names=('x',)); (x,) = K._first_ngens(1)
>>> R = K['y']; (y,) = R._first_ngens(1)
>>> L = K.extension(x - y**Integer(2), names=('w',)); (w,) = L._first_ngens(1)
↳# indirect doctest # needs sage.rings.function_field
>>> y2 = y*Integer(1)
>>> M = K.extension(x - y2**Integer(2), names=('w',)); (w,) = M._first_
↳ngens(1) # indirect doctest # needs sage.rings.function_
↳field
>>> L is M #_
↳needs sage.rings.function_field
True

```

class sage.rings.function_field.constructor.**FunctionFieldFactory**

Bases: UniqueFactory

Return the function field in one variable with constant field F . The function field returned is unique in the sense that if you call this function twice with the same base field and name then you get the same python object back.

INPUT:

- F – field
- names – name of variable as a string or a tuple containing a string

EXAMPLES:

```

sage: K.<x> = FunctionField(QQ); K
Rational function field in x over Rational Field
sage: L.<y> = FunctionField(GF(7)); L
Rational function field in y over Finite Field of size 7
sage: R.<z> = L[]
sage: M.<z> = L.extension(z^7 - z - y); M #_
↳needs sage.rings.finite_rings sage.rings.function_field
Function field in z defined by  $z^7 + 6z + 6y$ 

```

```

>>> from sage.all import *
>>> K = FunctionField(QQ, names=('x',)); (x,) = K._first_ngens(1); K
Rational function field in x over Rational Field
>>> L = FunctionField(GF(Integer(7)), names=('y',)); (y,) = L._first_ngens(1); L
Rational function field in y over Finite Field of size 7
>>> R = L['z']; (z,) = R._first_ngens(1)
>>> M = L.extension(z**Integer(7) - z - y, names=('z',)); (z,) = M._first_
↳ngens(1); M # needs sage.rings.finite_
↳rings sage.rings.function_field
Function field in z defined by  $z^7 + 6z + 6y$ 

```

create_key(*F*, *names*)

Given the arguments and keywords, create a key that uniquely determines this object.

EXAMPLES:

```
sage: K.<x> = FunctionField(QQ) # indirect doctest
```

```
>>> from sage.all import *
>>> K = FunctionField(QQ, names=('x',)); (x,) = K._first_ngens(1) # indirect_
↪doctest
```

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

Create the object from the key and extra arguments. This is only called if the object was not found in the cache.

EXAMPLES:

```
sage: K.<x> = FunctionField(QQ) # indirect doctest
sage: L.<x> = FunctionField(QQ) # indirect doctest
sage: K is L
True
```

```
>>> from sage.all import *
>>> K = FunctionField(QQ, names=('x',)); (x,) = K._first_ngens(1) # indirect_
↪doctest
>>> L = FunctionField(QQ, names=('x',)); (x,) = L._first_ngens(1) # indirect_
↪doctest
>>> K is L
True
```

A basic reference for the theory of algebraic function fields is [Stich2009].

JACOBIANS OF FUNCTION FIELDS

Arithmetic in Jacobians of function fields are available in two flavors.

26.1 Jacobians of function fields

This module provides base classes for Jacobians of function fields.

26.1.1 Jacobian

The Jacobian of a function field is created by default in the Hess model, with a base divisor of degree g the genus of the function field. The base divisor is automatically chosen if not given.

```
sage: P2.<x,y,z> = ProjectiveSpace(GF(29), 2)
sage: C = Curve(x^3 + 5*z^3 - y^2*z, P2)
sage: F = C.function_field()
sage: J = F.jacobian()
sage: J
Jacobian of Function field in z defined by z^3 + 23*y^2*z + 6 (Hess model)
sage: J.base_divisor().degree()
1
```

```
>>> from sage.all import *
>>> P2 = ProjectiveSpace(GF(Integer(29)), Integer(2), names=('x', 'y', 'z')); (x, y, z) = P2._first_ngens(3)
>>> C = Curve(x**Integer(3) + Integer(5)*z**Integer(3) - y**Integer(2)*z, P2)
>>> F = C.function_field()
>>> J = F.jacobian()
>>> J
Jacobian of Function field in z defined by z^3 + 23*y^2*z + 6 (Hess model)
>>> J.base_divisor().degree()
1
```

Explicitly specify a model if you want Jacobians in different models.

```
sage: J_km = F.jacobian(model='km_large')
sage: J_km
Jacobian of Function field in z defined by z^3 + 23*y^2*z + 6 (Khuri-Makdisi large model)
```

```
>>> from sage.all import *
>>> J_km = F.jacobian(model='km_large')
>>> J_km
Jacobian of Function field in z defined by z^3 + 23*y^2*z + 6 (Khuri-Makdisi large_
↪model)
```

26.1.2 Group of rational points

The group of rational points of a Jacobian is created from the Jacobian. A point of the Jacobian group is determined by a divisor of degree zero. To represent the point, a divisor of the form $D - B$ is selected where D is an effective divisor of the same degree with the base divisor B . Hence the point is simply represented by the divisor D .

```
sage: G = J.group()
sage: G.order()
30
sage: p11 = C([1,8,1]).place()
sage: p12 = C([2,10,1]).place()
sage: p1 = G.point(p11 - p12)
sage: p1
[Place (y + 1, z + 6)]
sage: p2 = G.point(p12 - p11)
sage: p2
[Place (y + 28, z + 6)]
sage: p1 + p2 == G.zero()
True
sage: p1.order()
5
```

```
>>> from sage.all import *
>>> G = J.group()
>>> G.order()
30
>>> p11 = C([Integer(1),Integer(8),Integer(1)]).place()
>>> p12 = C([Integer(2),Integer(10),Integer(1)]).place()
>>> p1 = G.point(p11 - p12)
>>> p1
[Place (y + 1, z + 6)]
>>> p2 = G.point(p12 - p11)
>>> p2
[Place (y + 28, z + 6)]
>>> p1 + p2 == G.zero()
True
>>> p1.order()
5
```

We can get the corresponding point in the Jacobian in a different model.

```
sage: # long time
sage: p1km = J_km(p1)
sage: p1km.order()
5
sage: p1km
```

(continues on next page)

(continued from previous page)

```
Point of Jacobian determined by
[ 1  0  0  0  0  0 11  0  0]
[ 0  1  0  0  0  0 18  0  0]
[ 0  0  1  0  0  0 11  0  0]
[ 0  0  0  1  0  0 18  1  0]
[ 0  0  0  0  1  0 25  0 19]
[ 0  0  0  0  0  1  8  8  0]
```

```
>>> from sage.all import *
>>> # long time
>>> p1km = J_km(p1)
>>> p1km.order()
5
>>> p1km
Point of Jacobian determined by
[ 1  0  0  0  0  0 11  0  0]
[ 0  1  0  0  0  0 18  0  0]
[ 0  0  1  0  0  0 11  0  0]
[ 0  0  0  1  0  0 18  1  0]
[ 0  0  0  0  1  0 25  0 19]
[ 0  0  0  0  0  1  8  8  0]
```

AUTHORS:

- Kwankyu Lee (2022-01-24): initial version

class sage.rings.function_field.jacobian_base.**JacobianGroupFunctor** (*base_field*, *field*)

Bases: `ConstructionFunctor`

A construction functor for Jacobian groups.

EXAMPLES:

```
sage: k = GF(7)
sage: P2.<x,y,z> = ProjectiveSpace(k, 2)
sage: C = Curve(x^3 + 5*z^3 - y^2*z, P2)
sage: J = C.jacobian(model='hess')
sage: G = J.group()
sage: F, obj = G.construction()
sage: F
JacobianGroupFunctor
```

```
>>> from sage.all import *
>>> k = GF(Integer(7))
>>> P2 = ProjectiveSpace(k, Integer(2), names=('x', 'y', 'z')); (x, y, z,) = P2._
↳first_ngens(3)
>>> C = Curve(x**Integer(3) + Integer(5)*z**Integer(3) - y**Integer(2)*z, P2)
>>> J = C.jacobian(model='hess')
>>> G = J.group()
>>> F, obj = G.construction()
>>> F
JacobianGroupFunctor
```

merge (*other*)

Return the functor merging `self` and `other`.

INPUT:

- `other` – a functor

EXAMPLES:

```
sage: k = GF(7)
sage: P2.<x,y,z> = ProjectiveSpace(k, 2)
sage: C = Curve(x^3 + 5*z^3 - y^2*z, P2)
sage: J = C.jacobian(model='hess')
sage: K2 = k.extension(2)
sage: G2 = J.group(K2)
sage: K3 = k.extension(3)
sage: G3 = J.group(K3)
sage: sage.categories.pushout.pushout(G2, G3) # indirect doctest
Group of rational points of Jacobian over Finite Field in z6 of size 7^6
↳ (Hess model)
```

```
>>> from sage.all import *
>>> k = GF(Integer(7))
>>> P2 = ProjectiveSpace(k, Integer(2), names=('x', 'y', 'z')); (x, y, z) = _
↳ P2._first_ngens(3)
>>> C = Curve(x**Integer(3) + Integer(5)*z**Integer(3) - y**Integer(2)*z, P2)
>>> J = C.jacobian(model='hess')
>>> K2 = k.extension(Integer(2))
>>> G2 = J.group(K2)
>>> K3 = k.extension(Integer(3))
>>> G3 = J.group(K3)
>>> sage.categories.pushout.pushout(G2, G3) # indirect doctest
Group of rational points of Jacobian over Finite Field in z6 of size 7^6
↳ (Hess model)
```

rank = 20

```
class sage.rings.function_field.jacobian_base.JacobianGroup_base(parent, function_field,
                                                                base_div)
```

Bases: `Parent`

Groups of rational points of Jacobians.

INPUT:

- `parent` – a Jacobian
- `function_field` – a function field
- `base_div` – an effective divisor of the function field

EXAMPLES:

```
sage: P2.<x,y,z> = ProjectiveSpace(GF(7), 2)
sage: C = Curve(x^3 + 5*z^3 - y^2*z, P2)
sage: J = C.jacobian(model='hess')
sage: J.group()
Group of rational points of Jacobian over Finite Field of size 7 (Hess model)
```

```

>>> from sage.all import *
>>> P2 = ProjectiveSpace(GF(Integer(7)), Integer(2), names=('x', 'y', 'z')); (x, y, z) = P2._first_ngens(3)
>>> C = Curve(x**Integer(3) + Integer(5)*z**Integer(3) - y**Integer(2)*z, P2)
>>> J = C.jacobian(model='hess')
>>> J.group()
Group of rational points of Jacobian over Finite Field of size 7 (Hess model)

```

base_divisor()

Return the base divisor that is used to represent points of this group.

EXAMPLES:

```

sage: P2.<x,y,z> = ProjectiveSpace(GF(7), 2)
sage: C = Curve(x^3 + 5*z^3 - y^2*z, P2)
sage: b = C([0,1,0]).place()
sage: J = C.jacobian(model='hess', base_div=b)
sage: G = J.group()
sage: G.base_divisor()
Place (1/y, 1/y*z)
sage: _ == 1*b
True

```

```

>>> from sage.all import *
>>> P2 = ProjectiveSpace(GF(Integer(7)), Integer(2), names=('x', 'y', 'z')); (x, y, z) = P2._first_ngens(3)
>>> C = Curve(x**Integer(3) + Integer(5)*z**Integer(3) - y**Integer(2)*z, P2)
>>> b = C([Integer(0), Integer(1), Integer(0)]).place()
>>> J = C.jacobian(model='hess', base_div=b)
>>> G = J.group()
>>> G.base_divisor()
Place (1/y, 1/y*z)
>>> _ == Integer(1)*b
True

```

The base divisor is the denominator (negative part) of the divisor of degree zero that represents a point.

```

sage: p = C([-1,2,1]).place()
sage: G.point(p - b).divisor()
- Place (1/y, 1/y*z)
+ Place (y + 2, z + 1)

```

```

>>> from sage.all import *
>>> p = C([-Integer(1), Integer(2), Integer(1)]).place()
>>> G.point(p - b).divisor()
- Place (1/y, 1/y*z)
+ Place (y + 2, z + 1)

```

construction()

Return the data for a functorial construction of this Jacobian group.

EXAMPLES:

```

sage: k = GF(7)
sage: P2.<x,y,z> = ProjectiveSpace(k, 2)
sage: C = Curve(x^3 + 5*z^3 - y^2*z, P2)
sage: J = C.jacobian(model='hess')
sage: K2 = k.extension(2)
sage: G2 = J.group(K2)
sage: K3 = k.extension(3)
sage: G3 = J.group(K3)
sage: p1, p2 = G2.get_points(2)
sage: q1, q2 = G3.get_points(2)
sage: (p1 + q1).parent() is (p2 + q2).parent()
True

```

```

>>> from sage.all import *
>>> k = GF(Integer(7))
>>> P2 = ProjectiveSpace(k, Integer(2), names=('x', 'y', 'z')); (x, y, z) = P2._first_ngens(3)
>>> C = Curve(x**Integer(3) + Integer(5)*z**Integer(3) - y**Integer(2)*z, P2)
>>> J = C.jacobian(model='hess')
>>> K2 = k.extension(Integer(2))
>>> G2 = J.group(K2)
>>> K3 = k.extension(Integer(3))
>>> G3 = J.group(K3)
>>> p1, p2 = G2.get_points(Integer(2))
>>> q1, q2 = G3.get_points(Integer(2))
>>> (p1 + q1).parent() is (p2 + q2).parent()
True

```

function_field()

Return the function field to which this Jacobian group attached.

EXAMPLES:

```

sage: P2.<x,y,z> = ProjectiveSpace(GF(7), 2)
sage: C = Curve(x^3 + 5*z^3 - y^2*z, P2)
sage: J = C.jacobian(model='hess')
sage: G = J.group()
sage: G.function_field()
Function field in z defined by z^3 + 4*y^2*z + 3

```

```

>>> from sage.all import *
>>> P2 = ProjectiveSpace(GF(Integer(7)), Integer(2), names=('x', 'y', 'z')); (x, y, z) = P2._first_ngens(3)
>>> C = Curve(x**Integer(3) + Integer(5)*z**Integer(3) - y**Integer(2)*z, P2)
>>> J = C.jacobian(model='hess')
>>> G = J.group()
>>> G.function_field()
Function field in z defined by z^3 + 4*y^2*z + 3

```

parent()

Return the Jacobian to which this Jacobian group belongs.

EXAMPLES:

```

sage: P2.<x,y,z> = ProjectiveSpace(GF(7), 2)
sage: C = Curve(x^3 + 5*z^3 - y^2*z, P2)
sage: J = C.jacobian(model='hess')
sage: G = J.group()
sage: G.parent()
Jacobian of Projective Plane Curve over Finite Field of size 7
defined by x^3 - y^2*z - 2*z^3 (Hess model)

```

```

>>> from sage.all import *
>>> P2 = ProjectiveSpace(GF(Integer(7)), Integer(2), names=('x', 'y', 'z'));
↪ (x, y, z) = P2._first_ngens(3)
>>> C = Curve(x**Integer(3) + Integer(5)*z**Integer(3) - y**Integer(2)*z, P2)
>>> J = C.jacobian(model='hess')
>>> G = J.group()
>>> G.parent()
Jacobian of Projective Plane Curve over Finite Field of size 7
defined by x^3 - y^2*z - 2*z^3 (Hess model)

```

```

class sage.rings.function_field.jacobian_base.JacobianGroup_finite_field_base(parent,
                                                                                   func-
                                                                                   tion_field,
                                                                                   base_div)

```

Bases: *JacobianGroup_base*

Jacobian groups of function fields over finite fields.

EXAMPLES:

```

sage: P2.<x,y,z> = ProjectiveSpace(GF(7), 2)
sage: C = Curve(x^3 + 5*z^3 - y^2*z, P2)
sage: b = C([0,1,0]).place()
sage: J = C.jacobian(model='hess', base_div=b)
sage: J.group()
Group of rational points of Jacobian over Finite Field of size 7 (Hess model)

```

```

>>> from sage.all import *
>>> P2 = ProjectiveSpace(GF(Integer(7)), Integer(2), names=('x', 'y', 'z')); (x, ↪
↪ y, z) = P2._first_ngens(3)
>>> C = Curve(x**Integer(3) + Integer(5)*z**Integer(3) - y**Integer(2)*z, P2)
>>> b = C([Integer(0), Integer(1), Integer(0)]).place()
>>> J = C.jacobian(model='hess', base_div=b)
>>> J.group()
Group of rational points of Jacobian over Finite Field of size 7 (Hess model)

```

get_points(*n*)

Return *n* points of the Jacobian group.

If *n* is greater than the order of the group, then returns all points of the group.

INPUT:

- *n* – integer

EXAMPLES:

```

sage: k = GF(7)
sage: A.<x,y> = AffineSpace(k,2)
sage: C = Curve(y^2 + x^3 + 2*x + 1).projective_closure()
sage: J = C.jacobian(model='hess')
sage: G = J.group()
sage: pts = G.get_points(G.order())
sage: len(pts)
11

```

```

>>> from sage.all import *
>>> k = GF(Integer(7))
>>> A = AffineSpace(k,Integer(2), names=('x', 'y',)); (x, y,) = A._first_
↳ngens(2)
>>> C = Curve(y**Integer(2) + x**Integer(3) + Integer(2)*x + Integer(1)).
↳projective_closure()
>>> J = C.jacobian(model='hess')
>>> G = J.group()
>>> pts = G.get_points(G.order())
>>> len(pts)
11

```

order (*algorithm*='numeric')

Return the order of the Jacobian group.

INPUT:

- *algorithm* - 'numeric' (default) or 'algebraic'

EXAMPLES:

```

sage: P2.<x,y,z> = ProjectiveSpace(GF(7), 2)
sage: C = Curve(x^3 + 5*z^3 - y^2*z, P2)
sage: b = C([0,1,0]).place()
sage: J = C.jacobian(model='hess', base_div=b)
sage: G = J.group()
sage: G.order()
7

```

```

>>> from sage.all import *
>>> P2 = ProjectiveSpace(GF(Integer(7)), Integer(2), names=('x', 'y', 'z',));
↳(x, y, z,) = P2._first_ngens(3)
>>> C = Curve(x**Integer(3) + Integer(5)*z**Integer(3) - y**Integer(2)*z, P2)
>>> b = C([Integer(0),Integer(1),Integer(0)]).place()
>>> J = C.jacobian(model='hess', base_div=b)
>>> G = J.group()
>>> G.order()
7

```

class sage.rings.function_field.jacobian_base.JacobianPoint_base

Bases: *ModuleElement*

Abstract base class of points of Jacobian groups.

class sage.rings.function_field.jacobian_base.JacobianPoint_finite_field_base

Bases: *JacobianPoint_base*

Points of Jacobians over finite fields.

frobenius()

Return the image of the point acted by the Frobenius automorphism.

EXAMPLES:

```
sage: k = GF(7)
sage: A.<x,y> = AffineSpace(k,2)
sage: C = Curve(y^2 + x^3 + 2*x + 1).projective_closure()
sage: J = C.jacobian(model='hess')
sage: G1 = J.group()
sage: G1.order()
11
sage: K = k.extension(3)
sage: G3 = J.group(K)
sage: pts1 = G1.get_points(11)
sage: pts3 = G3.get_points(12)
sage: pt = next(pt for pt in pts3 if pt not in pts1)
sage: pt.frobenius() == pt
False
sage: pt.frobenius().frobenius().frobenius() == pt
True
```

```
>>> from sage.all import *
>>> k = GF(Integer(7))
>>> A = AffineSpace(k,Integer(2), names=('x', 'y')); (x, y) = A._first_
↳ngens(2)
>>> C = Curve(y**Integer(2) + x**Integer(3) + Integer(2)*x + Integer(1)).
↳projective_closure()
>>> J = C.jacobian(model='hess')
>>> G1 = J.group()
>>> G1.order()
11
>>> K = k.extension(Integer(3))
>>> G3 = J.group(K)
>>> pts1 = G1.get_points(Integer(11))
>>> pts3 = G3.get_points(Integer(12))
>>> pt = next(pt for pt in pts3 if pt not in pts1)
>>> pt.frobenius() == pt
False
>>> pt.frobenius().frobenius().frobenius() == pt
True
```

order()

Return the order of this point.

EXAMPLES:

```
sage: # long time
sage: P2.<x,y,z> = ProjectiveSpace(GF(29), 2)
sage: C = Curve(x^3 + 5*z^3 - y^2*z, P2)
sage: F = C.function_field()
sage: h = C.function(y/x).divisor_of_poles()
```

(continues on next page)

(continued from previous page)

```

sage: J = C.jacobian(model='km_large', base_div=h)
sage: G = J.group()
sage: b = F.get_place(1)
sage: p1 = C([-1, 2, 1]).place()
sage: p = G.point(p1 - b)
sage: p.order()
15

```

```

>>> from sage.all import *
>>> # long time
>>> P2 = ProjectiveSpace(GF(Integer(29)), Integer(2), names=('x', 'y', 'z',));
↪ (x, y, z,) = P2._first_ngens(3)
>>> C = Curve(x**Integer(3) + Integer(5)*z**Integer(3) - y**Integer(2)*z, P2)
>>> F = C.function_field()
>>> h = C.function(y/x).divisor_of_poles()
>>> J = C.jacobian(model='km_large', base_div=h)
>>> G = J.group()
>>> b = F.get_place(Integer(1))
>>> p1 = C([-Integer(1), Integer(2), Integer(1)]).place()
>>> p = G.point(p1 - b)
>>> p.order()
15

```

ALGORITHM: Shanks' Baby Step Giant Step

class sage.rings.function_field.jacobian_base.**Jacobian_base**(*function_field*, *base_div*, ****kwds**)

Bases: `Parent`

Jacobians of function fields.

EXAMPLES:

```

sage: K.<x> = FunctionField(GF(2)); _.<Y> = K[]
sage: F.<y> = K.extension(Y^2 + Y + x + 1/x)
sage: F.jacobian()
Jacobian of Function field in y defined by y^2 + y + (x^2 + 1)/x (Hess model)

```

```

>>> from sage.all import *
>>> K = FunctionField(GF(Integer(2)), names=('x',)); (x,) = K._first_ngens(1); _↪
↪ = K['Y']; (Y,) = _._first_ngens(1)
>>> F = K.extension(Y**Integer(2) + Y + x + Integer(1)/x, names=('y',)); (y,) = F.↪
↪ _first_ngens(1)
>>> F.jacobian()
Jacobian of Function field in y defined by y^2 + y + (x^2 + 1)/x (Hess model)

```

base_curve()

Return the base curve or the function field that abstractly defines a curve.

EXAMPLES:

```

sage: K.<x> = FunctionField(GF(2)); _.<Y> = K[]
sage: F.<y> = K.extension(Y^2 + Y + x + 1/x)
sage: J = F.jacobian()

```

(continues on next page)

(continued from previous page)

```
sage: J.base_curve()
Function field in y defined by y^2 + y + (x^2 + 1)/x
```

```
>>> from sage.all import *
>>> K = FunctionField(GF(Integer(2)), names=('x',)); (x,) = K._first_ngens(1);
↪ _ = K['Y']; (Y,) = _._first_ngens(1)
>>> F = K.extension(Y**Integer(2) + Y + x + Integer(1)/x, names=('y',)); (y,) ↪
↪ = F._first_ngens(1)
>>> J = F.jacobian()
>>> J.base_curve()
Function field in y defined by y^2 + y + (x^2 + 1)/x
```

base_divisor()

Return the base divisor used to construct the Jacobian.

EXAMPLES:

```
sage: K.<x> = FunctionField(GF(2)); _.<Y> = K[]
sage: F.<y> = K.extension(Y^2 + Y + x + 1/x)
sage: b = F.get_place(1)
sage: J = F.jacobian(base_div=b)
sage: J.base_divisor() == b
True
```

```
>>> from sage.all import *
>>> K = FunctionField(GF(Integer(2)), names=('x',)); (x,) = K._first_ngens(1);
↪ _ = K['Y']; (Y,) = _._first_ngens(1)
>>> F = K.extension(Y**Integer(2) + Y + x + Integer(1)/x, names=('y',)); (y,) ↪
↪ = F._first_ngens(1)
>>> b = F.get_place(Integer(1))
>>> J = F.jacobian(base_div=b)
>>> J.base_divisor() == b
True
```

curve()

Return the projective curve to which this Jacobian is attached.

If the Jacobian was constructed from a function field, then returns nothing.

EXAMPLES:

```
sage: K.<x> = FunctionField(GF(2)); _.<Y> = K[]
sage: F.<y> = K.extension(Y^2 + Y + x + 1/x)
sage: J = F.jacobian()
sage: J.curve()
```

```
>>> from sage.all import *
>>> K = FunctionField(GF(Integer(2)), names=('x',)); (x,) = K._first_ngens(1);
↪ _ = K['Y']; (Y,) = _._first_ngens(1)
>>> F = K.extension(Y**Integer(2) + Y + x + Integer(1)/x, names=('y',)); (y,) ↪
↪ = F._first_ngens(1)
>>> J = F.jacobian()
>>> J.curve()
```

facade_for()

Return the system of groups that this Jacobian is a facade for.

The Jacobian can be seen as a facade for all groups of rational points over field extensions of the base constant field of the function field. This method returns only the internally constructed system of such groups.

EXAMPLES:

```
sage: K.<x> = FunctionField(GF(2)); _.<Y> = K[]
sage: F.<y> = K.extension(Y^2 + Y + x + 1/x)
sage: J = F.jacobian()
sage: J.facade_for()
[Group of rational points of Jacobian over Finite Field of size 2 (Hess
↪model)]
```

```
>>> from sage.all import *
>>> K = FunctionField(GF(Integer(2)), names=('x',)); (x,) = K._first_ngens(1);
↪ _ = K['Y']; (Y,) = _._first_ngens(1)
>>> F = K.extension(Y**Integer(2) + Y + x + Integer(1)/x, names=('y',)); (y,) ↪
↪ = F._first_ngens(1)
>>> J = F.jacobian()
>>> J.facade_for()
[Group of rational points of Jacobian over Finite Field of size 2 (Hess
↪model)]
```

group(k_ext=None)

Return the group of rational points of the Jacobian.

EXAMPLES:

```
sage: K.<x> = FunctionField(GF(2)); _.<Y> = K[]
sage: F.<y> = K.extension(Y^2 + Y + x + 1/x)
sage: b = F.get_place(1)
sage: J = F.jacobian(base_div=b)
sage: J.group()
Group of rational points of Jacobian over Finite Field of size 2 (Hess model)
```

```
>>> from sage.all import *
>>> K = FunctionField(GF(Integer(2)), names=('x',)); (x,) = K._first_ngens(1);
↪ _ = K['Y']; (Y,) = _._first_ngens(1)
>>> F = K.extension(Y**Integer(2) + Y + x + Integer(1)/x, names=('y',)); (y,) ↪
↪ = F._first_ngens(1)
>>> b = F.get_place(Integer(1))
>>> J = F.jacobian(base_div=b)
>>> J.group()
Group of rational points of Jacobian over Finite Field of size 2 (Hess model)
```

set_base_place(place)

Set place as the base place.

INPUT:

- *place* – a rational place of the function field

The base place B is used to map a rational place P of the function field to the point of the Jacobian defined by the divisor $P - B$.

EXAMPLES:

```

sage: K.<x> = FunctionField(GF(2)); _.<Y> = K[]
sage: F.<y> = K.extension(Y^2 + Y + x + 1/x)
sage: J = F.jacobian()
sage: B = F.get_place(1)
sage: J.set_base_place(B)
sage: Q = F.places()[-1]
sage: J(Q)
[Place (x + 1, x*y + 1)]
sage: J(Q).parent()
Group of rational points of Jacobian over Finite Field of size 2 (Hess model)
sage: J(B)
[Place (x, x*y)]
sage: J(B).is_zero()
True

```

```

>>> from sage.all import *
>>> K = FunctionField(GF(Integer(2)), names=('x',)); (x,) = K._first_ngens(1);
↪ _ = K['Y']; (Y,) = _._first_ngens(1)
>>> F = K.extension(Y**Integer(2) + Y + x + Integer(1)/x, names=('y',)); (y,) =
↪ F._first_ngens(1)
>>> J = F.jacobian()
>>> B = F.get_place(Integer(1))
>>> J.set_base_place(B)
>>> Q = F.places()[-Integer(1)]
>>> J(Q)
[Place (x + 1, x*y + 1)]
>>> J(Q).parent()
Group of rational points of Jacobian over Finite Field of size 2 (Hess model)
>>> J(B)
[Place (x, x*y)]
>>> J(B).is_zero()
True

```

26.2 Jacobians in Hess model

This module implements Jacobian arithmetic based on divisor representation by ideals. This approach to Jacobian arithmetic implementation is attributed to Hess [Hes2002].

26.2.1 Jacobian

To create a Jacobian in Hess model, specify 'hess' model and provide a base divisor of degree g , which is the genus of the function field:

```

sage: P2.<x,y,z> = ProjectiveSpace(GF(29), 2)
sage: C = Curve(x^3 + 5*z^3 - y^2*z, P2)
sage: C.geometric_genus()
1
sage: B = C([0,1,0]).place()
sage: B.degree()
1

```

(continues on next page)

(continued from previous page)

```
sage: J = C.jacobian(model='hess', base_div=B)
sage: J
Jacobian of Projective Plane Curve over Finite Field of size 29
defined by  $x^3 - y^2z + 5z^3$  (Hess model)
```

```
>>> from sage.all import *
>>> P2 = ProjectiveSpace(GF(Integer(29)), Integer(2), names=('x', 'y', 'z')); (x, y, z) = P2._first_ngens(3)
>>> C = Curve(x**Integer(3) + Integer(5)*z**Integer(3) - y**Integer(2)*z, P2)
>>> C.geometric_genus()
1
>>> B = C([Integer(0), Integer(1), Integer(0)]).place()
>>> B.degree()
1
>>> J = C.jacobian(model='hess', base_div=B)
>>> J
Jacobian of Projective Plane Curve over Finite Field of size 29
defined by  $x^3 - y^2z + 5z^3$  (Hess model)
```

26.2.2 Group of rational points

The group of rational points of a Jacobian is created from the Jacobian. A point of the Jacobian group is determined by a divisor (of degree zero) of the form $D - B$ where D is an effective divisor of degree g and B is the base divisor. Hence a point of the Jacobian group is represented by D .

```
sage: G = J.group()
sage: P1 = C([1, 8, 1]).place()
sage: P2 = C([2, 10, 1]).place()
sage: p1 = G(P1)
sage: p2 = G(P2)
sage: p1
[Place (y + 21, z + 28)]
sage: p2
[Place (y + 24, z + 14)]
sage: p1 + p2
[Place (y + 8, z + 28)]
```

```
>>> from sage.all import *
>>> G = J.group()
>>> P1 = C([Integer(1), Integer(8), Integer(1)]).place()
>>> P2 = C([Integer(2), Integer(10), Integer(1)]).place()
>>> p1 = G(P1)
>>> p2 = G(P2)
>>> p1
[Place (y + 21, z + 28)]
>>> p2
[Place (y + 24, z + 14)]
>>> p1 + p2
[Place (y + 8, z + 28)]
```

AUTHORS:

- Kwankyu Lee (2022-01-24): initial version

```
class sage.rings.function_field.jacobian_hess.Jacobian(function_field, base_div, **kws)
```

Bases: *Jacobian_base*, *UniqueRepresentation*

Jacobians of function fields.

EXAMPLES:

```
sage: k = GF(17)
sage: P2.<x,y,z> = ProjectiveSpace(k, 2)
sage: C = Curve(x^3 + 5*z^3 - y^2*z, P2)
sage: b = C([0,1,0]).place()
sage: C.jacobian(model='hess', base_div=b)
Jacobian of Projective Plane Curve over Finite Field of size 17
defined by x^3 - y^2*z + 5*z^3 (Hess model)
```

```
>>> from sage.all import *
>>> k = GF(Integer(17))
>>> P2 = ProjectiveSpace(k, Integer(2), names=('x', 'y', 'z')); (x, y, z,) = P2._
↪ first_ngens(3)
>>> C = Curve(x**Integer(3) + Integer(5)*z**Integer(3) - y**Integer(2)*z, P2)
>>> b = C([Integer(0), Integer(1), Integer(0)]).place()
>>> C.jacobian(model='hess', base_div=b)
Jacobian of Projective Plane Curve over Finite Field of size 17
defined by x^3 - y^2*z + 5*z^3 (Hess model)
```

```
class sage.rings.function_field.jacobian_hess.JacobianGroup(parent, function_field, base_div)
```

Bases: *UniqueRepresentation*, *JacobianGroup_base*

Groups of rational points of a Jacobian.

INPUT:

- parent – a Jacobian
- function_field – a function field
- base_div – an effective divisor of the function field

EXAMPLES:

```
sage: P2.<x,y,z> = ProjectiveSpace(GF(17), 2)
sage: C = Curve(x^3 + 5*z^3 - y^2*z, P2)
sage: b = C([0,1,0]).place()
sage: J = C.jacobian(model='hess', base_div=b)
sage: J.group()
Group of rational points of Jacobian
over Finite Field of size 17 (Hess model)
```

```
>>> from sage.all import *
>>> P2 = ProjectiveSpace(GF(Integer(17)), Integer(2), names=('x', 'y', 'z')); (x,
↪ y, z,) = P2._first_ngens(3)
>>> C = Curve(x**Integer(3) + Integer(5)*z**Integer(3) - y**Integer(2)*z, P2)
>>> b = C([Integer(0), Integer(1), Integer(0)]).place()
>>> J = C.jacobian(model='hess', base_div=b)
>>> J.group()
Group of rational points of Jacobian
over Finite Field of size 17 (Hess model)
```

Elementalias of *JacobianPoint***point** (*divisor*)

Return the point represented by the divisor of degree zero.

EXAMPLES:

```
sage: P2.<x,y,z> = ProjectiveSpace(GF(17), 2)
sage: C = Curve(x^3 + 5*z^3 - y^2*z, P2)
sage: b = C([0,1,0]).place()
sage: J = C.jacobian(model='hess', base_div=b)
sage: G = J.group()
sage: p = C([-1,2,1]).place()
sage: G.point(p - b)
[Place (y + 2, z + 1)]
```

```
>>> from sage.all import *
>>> P2 = ProjectiveSpace(GF(Integer(17)), Integer(2), names=('x', 'y', 'z'));
↪ (x, y, z,) = P2._first_ngens(3)
>>> C = Curve(x**Integer(3) + Integer(5)*z**Integer(3) - y**Integer(2)*z, P2)
>>> b = C([Integer(0),Integer(1),Integer(0)]).place()
>>> J = C.jacobian(model='hess', base_div=b)
>>> G = J.group()
>>> p = C([-Integer(1),Integer(2),Integer(1)]).place()
>>> G.point(p - b)
[Place (y + 2, z + 1)]
```

zero ()

Return the zero element of this group.

EXAMPLES:

```
sage: P2.<x,y,z> = ProjectiveSpace(GF(17), 2)
sage: C = Curve(x^3 + 5*z^3 - y^2*z, P2)
sage: b = C([0,1,0]).place()
sage: J = C.jacobian(model='hess', base_div=b)
sage: G = J.group()
sage: G.zero()
[Place (1/y, 1/y*z)]
```

```
>>> from sage.all import *
>>> P2 = ProjectiveSpace(GF(Integer(17)), Integer(2), names=('x', 'y', 'z'));
↪ (x, y, z,) = P2._first_ngens(3)
>>> C = Curve(x**Integer(3) + Integer(5)*z**Integer(3) - y**Integer(2)*z, P2)
>>> b = C([Integer(0),Integer(1),Integer(0)]).place()
>>> J = C.jacobian(model='hess', base_div=b)
>>> G = J.group()
>>> G.zero()
[Place (1/y, 1/y*z)]
```

```
class sage.rings.function_field.jacobian_hess.JacobianGroupEmbedding (base_group,
                                                                    extension_group)
```

Bases: Map

Embeddings between Jacobian groups.

INPUT:

- `base_group` – Jacobian group over a base field
- `extension_group` – Jacobian group over an extension field

EXAMPLES:

```
sage: k = GF(17)
sage: P2.<x,y,z> = ProjectiveSpace(k, 2)
sage: C = Curve(x^3 + 5*z^3 - y^2*z, P2)
sage: b = C([0,1,0]).place()
sage: J = C.jacobian(model='hess', base_div=b)
sage: G1 = J.group()
sage: K = k.extension(3)
sage: G3 = J.group(K)
sage: G3.coerce_map_from(G1)
Jacobian group embedding map:
  From: Group of rational points of Jacobian
        over Finite Field of size 17 (Hess model)
  To:   Group of rational points of Jacobian
        over Finite Field in z3 of size 17^3 (Hess model)
```

```
>>> from sage.all import *
>>> k = GF(Integer(17))
>>> P2 = ProjectiveSpace(k, Integer(2), names=('x', 'y', 'z')); (x, y, z,) = P2._
↳first_ngens(3)
>>> C = Curve(x**Integer(3) + Integer(5)*z**Integer(3) - y**Integer(2)*z, P2)
>>> b = C([Integer(0),Integer(1),Integer(0)]).place()
>>> J = C.jacobian(model='hess', base_div=b)
>>> G1 = J.group()
>>> K = k.extension(Integer(3))
>>> G3 = J.group(K)
>>> G3.coerce_map_from(G1)
Jacobian group embedding map:
  From: Group of rational points of Jacobian
        over Finite Field of size 17 (Hess model)
  To:   Group of rational points of Jacobian
        over Finite Field in z3 of size 17^3 (Hess model)
```

```
class sage.rings.function_field.jacobian_hess.JacobianGroup_finite_field(parent,
                                                                           function_field,
                                                                           base_div)
```

Bases: *JacobianGroup*, *JacobianGroup_finite_field_base*

Jacobian groups of function fields over finite fields.

INPUT:

- `parent` – a Jacobian
- `function_field` – a function field
- `base_div` – an effective divisor of the function field

EXAMPLES:

```

sage: k = GF(17)
sage: P2.<x,y,z> = ProjectiveSpace(k, 2)
sage: C = Curve(x^3 + 5*z^3 - y^2*z, P2)
sage: b = C([0,1,0]).place()
sage: J = C.jacobian(model='hess', base_div=b)
sage: G1 = J.group()
sage: K = k.extension(3)
sage: G3 = J.group(K)
sage: G3.coerce_map_from(G1)
Jacobian group embedding map:
  From: Group of rational points of Jacobian
        over Finite Field of size 17 (Hess model)
  To:   Group of rational points of Jacobian
        over Finite Field in z3 of size 17^3 (Hess model)

```

```

>>> from sage.all import *
>>> k = GF(Integer(17))
>>> P2 = ProjectiveSpace(k, Integer(2), names=('x', 'y', 'z')); (x, y, z,) = P2._
↳first_ngens(3)
>>> C = Curve(x**Integer(3) + Integer(5)*z**Integer(3) - y**Integer(2)*z, P2)
>>> b = C([Integer(0),Integer(1),Integer(0)]).place()
>>> J = C.jacobian(model='hess', base_div=b)
>>> G1 = J.group()
>>> K = k.extension(Integer(3))
>>> G3 = J.group(K)
>>> G3.coerce_map_from(G1)
Jacobian group embedding map:
  From: Group of rational points of Jacobian
        over Finite Field of size 17 (Hess model)
  To:   Group of rational points of Jacobian
        over Finite Field in z3 of size 17^3 (Hess model)

```

Elementalias of *JacobianPoint_finite_field***class** sage.rings.function_field.jacobian_hess.**JacobianPoint** (*parent, dS, ds*)Bases: *JacobianPoint_base*

Points of Jacobians represented by a pair of ideals.

If a point of Jacobian is determined by D , then the divisor D is represented by a pair of ideals in the finite maximal order and the infinite maximal order of the function field.

For efficiency reasons, the actual ideals stored are the inverted ideals of the ideals representing the divisor D .

INPUT:

- *parent* – Jacobian group
- *dS* – an ideal of the finite maximal order of a function field
- *ds* – an ideal of infinite maximal order of a function field

EXAMPLES:

```

sage: P2.<x,y,z> = ProjectiveSpace(GF(29), 2)
sage: C = Curve(x^3 + 5*z^3 - y^2*z, P2)

```

(continues on next page)

(continued from previous page)

```

sage: b = C([0,1,0]).place()
sage: G = C.jacobian(model='hess', base_div=b).group()
sage: p1 = C([1,8,1]).place()
sage: p = G.point(p1 - b)
sage: dS, ds = p._data
sage: -(dS.divisor() + ds.divisor()) == p1
True

```

```

>>> from sage.all import *
>>> P2 = ProjectiveSpace(GF(Integer(29)), Integer(2), names=('x', 'y', 'z')); (x,
↪ y, z) = P2._first_ngens(3)
>>> C = Curve(x**Integer(3) + Integer(5)*z**Integer(3) - y**Integer(2)*z, P2)
>>> b = C([Integer(0), Integer(1), Integer(0)]).place()
>>> G = C.jacobian(model='hess', base_div=b).group()
>>> p1 = C([Integer(1), Integer(8), Integer(1)]).place()
>>> p = G.point(p1 - b)
>>> dS, ds = p._data
>>> -(dS.divisor() + ds.divisor()) == p1
True

```

addflip(*other*)

Return the addflip of this and other point.

EXAMPLES:

```

sage: P2.<x,y,z> = ProjectiveSpace(GF(29), 2)
sage: C = Curve(x^3 + 5*z^3 - y^2*z, P2)
sage: b = C([0,1,0]).place()
sage: G = C.jacobian(model='hess', base_div=b).group()
sage: p11 = C([-1,2,1]).place()
sage: p12 = C([2,19,1]).place()
sage: p1 = G.point(p11 - b)
sage: p2 = G.point(p12 - b)
sage: p1.addflip(p2)
[Place (y + 8, z + 27)]
sage: _ == -(p1 + p2)
True

```

```

>>> from sage.all import *
>>> P2 = ProjectiveSpace(GF(Integer(29)), Integer(2), names=('x', 'y', 'z'));
↪ (x, y, z) = P2._first_ngens(3)
>>> C = Curve(x**Integer(3) + Integer(5)*z**Integer(3) - y**Integer(2)*z, P2)
>>> b = C([Integer(0), Integer(1), Integer(0)]).place()
>>> G = C.jacobian(model='hess', base_div=b).group()
>>> p11 = C([-Integer(1), Integer(2), Integer(1)]).place()
>>> p12 = C([Integer(2), Integer(19), Integer(1)]).place()
>>> p1 = G.point(p11 - b)
>>> p2 = G.point(p12 - b)
>>> p1.addflip(p2)
[Place (y + 8, z + 27)]
>>> _ == -(p1 + p2)
True

```

defining_divisor()

Return the effective divisor that defines this point.

EXAMPLES:

```
sage: P2.<x,y,z> = ProjectiveSpace(GF(29), 2)
sage: C = Curve(x^3 + 5*z^3 - y^2*z, P2)
sage: b = C([0,1,0]).place()
sage: G = C.jacobian(model='hess', base_div=b).group()
sage: p1 = C([-1,2,1]).place()
sage: p = G.point(p1 - b)
sage: p.defining_divisor() == p1
True
```

```
>>> from sage.all import *
>>> P2 = ProjectiveSpace(GF(Integer(29)), Integer(2), names=('x', 'y', 'z'));
↪ (x, y, z) = P2._first_ngens(3)
>>> C = Curve(x**Integer(3) + Integer(5)*z**Integer(3) - y**Integer(2)*z, P2)
>>> b = C([Integer(0),Integer(1),Integer(0)]).place()
>>> G = C.jacobian(model='hess', base_div=b).group()
>>> p1 = C([-Integer(1),Integer(2),Integer(1)]).place()
>>> p = G.point(p1 - b)
>>> p.defining_divisor() == p1
True
```

divisor()

Return the divisor representing this point.

EXAMPLES:

```
sage: P2.<x,y,z> = ProjectiveSpace(GF(29), 2)
sage: C = Curve(x^3 + 5*z^3 - y^2*z, P2)
sage: b = C([0,1,0]).place()
sage: G = C.jacobian(model='hess', base_div=b).group()
sage: p1 = C([-1,2,1]).place()
sage: p = G.point(p1 - b)
sage: G.point(p.divisor()) == p
True
```

```
>>> from sage.all import *
>>> P2 = ProjectiveSpace(GF(Integer(29)), Integer(2), names=('x', 'y', 'z'));
↪ (x, y, z) = P2._first_ngens(3)
>>> C = Curve(x**Integer(3) + Integer(5)*z**Integer(3) - y**Integer(2)*z, P2)
>>> b = C([Integer(0),Integer(1),Integer(0)]).place()
>>> G = C.jacobian(model='hess', base_div=b).group()
>>> p1 = C([-Integer(1),Integer(2),Integer(1)]).place()
>>> p = G.point(p1 - b)
>>> G.point(p.divisor()) == p
True
```

multiple(n)

Return the n -th multiple of this point.

EXAMPLES:

```

sage: P2.<x,y,z> = ProjectiveSpace(GF(29), 2)
sage: C = Curve(x^3 + 5*z^3 - y^2*z, P2)
sage: b = C([0,1,0]).place()
sage: G = C.jacobian(model='hess', base_div=b).group()
sage: p1 = C([-1,2,1]).place()
sage: p = G.point(p1 - b)
sage: p.multiple(100)
[Place (1/y, 1/y*z + 8)]

```

```

>>> from sage.all import *
>>> P2 = ProjectiveSpace(GF(Integer(29)), Integer(2), names=('x', 'y', 'z'));
↪ (x, y, z,) = P2._first_ngens(3)
>>> C = Curve(x**Integer(3) + Integer(5)*z**Integer(3) - y**Integer(2)*z, P2)
>>> b = C([Integer(0),Integer(1),Integer(0)]).place()
>>> G = C.jacobian(model='hess', base_div=b).group()
>>> p1 = C([-Integer(1),Integer(2),Integer(1)]).place()
>>> p = G.point(p1 - b)
>>> p.multiple(Integer(100))
[Place (1/y, 1/y*z + 8)]

```

order (*bound=None*)

Return the order of this point.

ALGORITHM: Shanks' Baby Step Giant Step

EXAMPLES:

```

sage: P2.<x,y,z> = ProjectiveSpace(GF(29), 2)
sage: C = Curve(x^3 + 5*z^3 - y^2*z, P2)
sage: b = C([0,1,0]).place()
sage: G = C.jacobian(model='hess', base_div=b).group()
sage: p = C([-1,2,1]).place()
sage: pt = G.point(p - b)
sage: pt.order()
30

```

```

>>> from sage.all import *
>>> P2 = ProjectiveSpace(GF(Integer(29)), Integer(2), names=('x', 'y', 'z'));
↪ (x, y, z,) = P2._first_ngens(3)
>>> C = Curve(x**Integer(3) + Integer(5)*z**Integer(3) - y**Integer(2)*z, P2)
>>> b = C([Integer(0),Integer(1),Integer(0)]).place()
>>> G = C.jacobian(model='hess', base_div=b).group()
>>> p = C([-Integer(1),Integer(2),Integer(1)]).place()
>>> pt = G.point(p - b)
>>> pt.order()
30

```

class sage.rings.function_field.jacobian_hess.JacobianPoint_finite_field(*parent, dS, ds*)

Bases: *JacobianPoint, JacobianPoint_finite_field_base*

Points of Jacobians over finite fields

26.3 Jacobians in Khuri-Makdisi model

This module implements Jacobian arithmetic by Khuri-Makdisi's algorithms [Khu2004] based on divisor representation by linear spaces.

26.3.1 Jacobian

There are three models for Jacobian arithmetic by Khuri-Makdisi's algorithms. For each of the models, one should provide a base divisor satisfying certain degree condition. The following lists the names of the three models and the corresponding conditions on base divisors. Let g be the genus of the function field.

- `km_large`: large model; requires an effective divisor of degree at least $2g + 1$
- `km_medium`: medium model; requires an effective divisor of degree at least $2g + 1$
- `km_small`: small model; requires an effective divisor of degree at least $g + 1$

To create a Jacobian in this model, specify '`km_[large|medium|small]`' as `model` and provide a base divisor satisfying the degree condition.

EXAMPLES:

We construct a function field (of a projective curve) over a finite field:

```
sage: P2.<x,y,z> = ProjectiveSpace(GF(29), 2)
sage: C = Curve(x^3 + 5*z^3 - y^2*z, P2)
sage: C.geometric_genus()
1
sage: H = C.function(y/x).divisor_of_poles()
sage: H.degree()
3
```

```
>>> from sage.all import *
>>> P2 = ProjectiveSpace(GF(Integer(29)), Integer(2), names=('x', 'y', 'z')); (x, y, z) = P2._first_ngens(3)
>>> C = Curve(x**Integer(3) + Integer(5)*z**Integer(3) - y**Integer(2)*z, P2)
>>> C.geometric_genus()
1
>>> H = C.function(y/x).divisor_of_poles()
>>> H.degree()
3
```

Now we use H as base divisor for the large and medium models:

```
sage: J_large = C.jacobian(model='km_large', base_div=H)
sage: J_large
Jacobian of Projective Plane Curve over Finite Field of size 29
defined by x^3 - y^2*z + 5*z^3 (Khuri-Makdisi large model)
sage: J_medium = C.jacobian(model='km_medium', base_div=H)
sage: J_medium
Jacobian of Projective Plane Curve over Finite Field of size 29
defined by x^3 - y^2*z + 5*z^3 (Khuri-Makdisi medium model)
```

```
>>> from sage.all import *
>>> J_large = C.jacobian(model='km_large', base_div=H)
>>> J_large
```

(continues on next page)

(continued from previous page)

```
Jacobian of Projective Plane Curve over Finite Field of size 29
defined by x^3 - y^2*z + 5*z^3 (Khuri-Makdisi large model)
>>> J_medium = C.jacobian(model='km_medium', base_div=H)
>>> J_medium
Jacobian of Projective Plane Curve over Finite Field of size 29
defined by x^3 - y^2*z + 5*z^3 (Khuri-Makdisi medium model)
```

and for the small model, we construct an effective divisor of degree 2:

```
sage: B = sum(H.support()[0:2])
sage: B.degree()
2
sage: J_small = C.jacobian(model='km_small', base_div=B)
sage: J_small
Jacobian of Projective Plane Curve over Finite Field of size 29
defined by x^3 - y^2*z + 5*z^3 (Khuri-Makdisi small model)
```

```
>>> from sage.all import *
>>> B = sum(H.support()[0:Integer(2)])
>>> B.degree()
2
>>> J_small = C.jacobian(model='km_small', base_div=B)
>>> J_small
Jacobian of Projective Plane Curve over Finite Field of size 29
defined by x^3 - y^2*z + 5*z^3 (Khuri-Makdisi small model)
```

26.3.2 Group of rational points

The group of rational points of a Jacobian is created from the Jacobian. A point of the Jacobian group is represented by a divisor $D - B$ where D is an effective divisor of the same degree with the base divisor B . The divisor D in turn is determined by a linear subspace of the Riemann-Roch space associated with certain multiple of B (depending on the model). This allows representing points of Jacobian as matrices once we fix a basis of the Riemann-Roch space.

EXAMPLES:

```
sage: # long time
sage: P2.<x,y,z> = ProjectiveSpace(GF(17), 2)
sage: C = Curve(x^3 + 5*z^3 - y^2*z, P2)
sage: F = C.function_field()
sage: H = C.function(y/x).divisor_of_poles()
sage: J = C.jacobian(model='km_large', base_div=H)
sage: G = J.group()
sage: D = C([0,1,0]).place()
sage: P1 = C([-1,2,1]).place()
sage: P2 = C([3,7,1]).place()
sage: p1 = G.point(P1 - D)
sage: p2 = G.point(P2 - D)
sage: p1
Point of Jacobian determined by
[ 1  0  0  0  0  0  0 12 15]
[ 0  1  0  0  0  0  0  0 13]
[ 0  0  1  0  0  0  0  0  2]
```

(continues on next page)

(continued from previous page)

```

[ 0 0 0 1 0 0 0 0 16]
[ 0 0 0 0 0 1 0 0 15]
[ 0 0 0 0 0 0 1 0 1]
sage: p2
Point of Jacobian determined by
[ 1 0 0 0 0 0 0 12 5]
[ 0 1 0 0 0 0 0 0 2]
[ 0 0 1 0 0 0 0 0 13]
[ 0 0 0 1 0 0 0 0 8]
[ 0 0 0 0 0 1 0 0 10]
[ 0 0 0 0 0 0 1 0 14]
sage: p1 + p2
Point of Jacobian determined by
[ 1 0 0 0 0 16 0 5 3]
[ 0 1 0 0 0 6 0 8 16]
[ 0 0 1 0 0 15 0 3 10]
[ 0 0 0 1 0 3 0 0 0]
[ 0 0 0 0 1 12 0 16 8]
[ 0 0 0 0 0 0 1 3 0]

```

```

>>> from sage.all import *
>>> # long time
>>> P2 = ProjectiveSpace(GF(Integer(17)), Integer(2), names=('x', 'y', 'z')); (x, y, z) = P2._first_ngens(3)
>>> C = Curve(x**Integer(3) + Integer(5)*z**Integer(3) - y**Integer(2)*z, P2)
>>> F = C.function_field()
>>> H = C.function(y/x).divisor_of_poles()
>>> J = C.jacobian(model='km_large', base_div=H)
>>> G = J.group()
>>> D = C([Integer(0), Integer(1), Integer(0)]).place()
>>> P1 = C([-Integer(1), Integer(2), Integer(1)]).place()
>>> P2 = C([Integer(3), Integer(7), Integer(1)]).place()
>>> p1 = G.point(P1 - D)
>>> p2 = G.point(P2 - D)
>>> p1
Point of Jacobian determined by
[ 1 0 0 0 0 0 0 12 15]
[ 0 1 0 0 0 0 0 0 13]
[ 0 0 1 0 0 0 0 0 2]
[ 0 0 0 1 0 0 0 0 16]
[ 0 0 0 0 0 1 0 0 15]
[ 0 0 0 0 0 0 1 0 1]
>>> p2
Point of Jacobian determined by
[ 1 0 0 0 0 0 0 12 5]
[ 0 1 0 0 0 0 0 0 2]
[ 0 0 1 0 0 0 0 0 13]
[ 0 0 0 1 0 0 0 0 8]
[ 0 0 0 0 0 1 0 0 10]
[ 0 0 0 0 0 0 1 0 14]
>>> p1 + p2
Point of Jacobian determined by

```

(continues on next page)

(continued from previous page)

```
[ 1  0  0  0  0 16  0  5  3]
[ 0  1  0  0  0  6  0  8 16]
[ 0  0  1  0  0 15  0  3 10]
[ 0  0  0  1  0  3  0  0  0]
[ 0  0  0  0  1 12  0 16  8]
[ 0  0  0  0  0  0  1  3  0]
```

AUTHORS:

- Kwankyu Lee (2022-01-24): initial version

```
class sage.rings.function_field.jacobian_khuri_makdisi.Jacobian(function_field, base_div,
                                                                model, **kwds)
```

Bases: `UniqueRepresentation`, `Jacobian_base`

Jacobians implemented by Khuri-Makdisi's algorithms.

EXAMPLES:

```
sage: P2.<x,y,z> = ProjectiveSpace(GF(7), 2)
sage: C = Curve(x^3 + 5*z^3 - y^2*z, P2)
sage: C.jacobian(model='km')
Jacobian of Projective Plane Curve over Finite Field of size 7
defined by x^3 - y^2*z - 2*z^3 (Khuri-Makdisi large model)
```

```
>>> from sage.all import *
>>> P2 = ProjectiveSpace(GF(Integer(7)), Integer(2), names=('x', 'y', 'z')); (x, y, z) = P2._first_ngens(3)
>>> C = Curve(x**Integer(3) + Integer(5)*z**Integer(3) - y**Integer(2)*z, P2)
>>> C.jacobian(model='km')
Jacobian of Projective Plane Curve over Finite Field of size 7
defined by x^3 - y^2*z - 2*z^3 (Khuri-Makdisi large model)
```

```
class sage.rings.function_field.jacobian_khuri_makdisi.JacobianGroup(parent, function_field,
                                                                    base_div)
```

Bases: `UniqueRepresentation`, `JacobianGroup_base`

Groups of rational points of a Jacobian.

INPUT:

- `parent` – a Jacobian
- `function_field` – a function field
- `base_div` – an effective divisor of the function field

EXAMPLES:

```
sage: # long time
sage: P2.<x,y,z> = ProjectiveSpace(GF(7), 2)
sage: C = Curve(x^3 + 5*z^3 - y^2*z, P2)
sage: h = C.function(y/x).divisor_of_poles()
sage: J = C.jacobian(model='km_large', base_div=h)
sage: J.group()
Group of rational points of Jacobian
over Finite Field of size 7 (Khuri-Makdisi large model)
```

```

>>> from sage.all import *
>>> # long time
>>> P2 = ProjectiveSpace(GF(Integer(7)), Integer(2), names=('x', 'y', 'z')); (x, y, z) = P2._first_ngens(3)
>>> C = Curve(x**Integer(3) + Integer(5)*z**Integer(3) - y**Integer(2)*z, P2)
>>> h = C.function(y/x).divisor_of_poles()
>>> J = C.jacobian(model='km_large', base_div=h)
>>> J.group()
Group of rational points of Jacobian
over Finite Field of size 7 (Khuri-Makdisi large model)

```

Element

alias of *JacobianPoint*

point (divisor)

Return the point represented by the divisor.

INPUT:

- divisor – a divisor of degree zero

EXAMPLES:

```

sage: # long time
sage: P2.<x,y,z> = ProjectiveSpace(GF(7), 2)
sage: C = Curve(x^3 + 5*z^3 - y^2*z, P2)
sage: h = C.function(y/x).divisor_of_poles()
sage: J = C.jacobian(model='km_large', base_div=h)
sage: G = J.group()
sage: b = C([0,1,0]).place()
sage: p = C([-1,2,1]).place()
sage: G.point(p - b)
Point of Jacobian determined by
[1 0 0 0 0 0 2 5]
[0 1 0 0 0 0 0 3]
[0 0 1 0 0 0 0 2]
[0 0 0 1 0 0 0 6]
[0 0 0 0 0 1 0 5]
[0 0 0 0 0 0 1 0 1]

```

```

>>> from sage.all import *
>>> # long time
>>> P2 = ProjectiveSpace(GF(Integer(7)), Integer(2), names=('x', 'y', 'z')); (x, y, z) = P2._first_ngens(3)
>>> C = Curve(x**Integer(3) + Integer(5)*z**Integer(3) - y**Integer(2)*z, P2)
>>> h = C.function(y/x).divisor_of_poles()
>>> J = C.jacobian(model='km_large', base_div=h)
>>> G = J.group()
>>> b = C([Integer(0),Integer(1),Integer(0)]).place()
>>> p = C([-Integer(1),Integer(2),Integer(1)]).place()
>>> G.point(p - b)
Point of Jacobian determined by
[1 0 0 0 0 0 2 5]
[0 1 0 0 0 0 0 3]

```

(continues on next page)

(continued from previous page)

```
[0 0 1 0 0 0 0 0 2]
[0 0 0 1 0 0 0 0 6]
[0 0 0 0 0 1 0 0 5]
[0 0 0 0 0 0 1 0 1]
```

zero()

Return the zero element of this group.

EXAMPLES:

```
sage: # long time
sage: P2.<x,y,z> = ProjectiveSpace(GF(7), 2)
sage: C = Curve(x^3 + 5*z^3 - y^2*z, P2)
sage: h = C.function(y/x).divisor_of_poles()
sage: J = C.jacobian(model='km_large', base_div=h)
sage: G = J.group()
sage: G.zero()
```

Point of Jacobian determined by

```
[1 0 0 0 0 0 0 0 0]
[0 1 0 0 0 0 0 0 0]
[0 0 1 0 0 0 0 0 0]
[0 0 0 0 1 0 0 0 0]
[0 0 0 0 0 1 0 0 0]
[0 0 0 0 0 0 1 0 0]
[0 0 0 0 0 0 0 1 0]
```

```
>>> from sage.all import *
>>> # long time
>>> P2 = ProjectiveSpace(GF(Integer(7)), Integer(2), names=('x', 'y', 'z'));
↳ (x, y, z,) = P2._first_ngens(3)
>>> C = Curve(x**Integer(3) + Integer(5)*z**Integer(3) - y**Integer(2)*z, P2)
>>> h = C.function(y/x).divisor_of_poles()
>>> J = C.jacobian(model='km_large', base_div=h)
>>> G = J.group()
>>> G.zero()
Point of Jacobian determined by
[1 0 0 0 0 0 0 0 0]
[0 1 0 0 0 0 0 0 0]
[0 0 1 0 0 0 0 0 0]
[0 0 0 0 1 0 0 0 0]
[0 0 0 0 0 1 0 0 0]
[0 0 0 0 0 0 1 0 0]
[0 0 0 0 0 0 0 1 0]
```

```
class sage.rings.function_field.jacobian_khuri_makdisi.JacobianGroupEmbedding(base_group,
                                                                                   extension_group)
```

Bases: [Map](#)

Embeddings between Jacobian groups.

INPUT:

- `base_group` – Jacobian group over a base field
- `extension_group` – Jacobian group over an extension field

EXAMPLES:

```
sage: # long time
sage: k = GF(5)
sage: P2.<x,y,z> = ProjectiveSpace(k, 2)
sage: C = Curve(x^3 + z^3 - y^2*z, P2)
sage: h = C.function(y/x).divisor_of_poles()
sage: J = C.jacobian(model='km_large', base_div=h)
sage: G1 = J.group()
sage: K = k.extension(2)
sage: G2 = J.group(K)
sage: G2.coerce_map_from(G1)
Jacobian group embedding map:
  From: Group of rational points of Jacobian
        over Finite Field of size 5 (Khuri-Makdisi large model)
  To:   Group of rational points of Jacobian
        over Finite Field in z2 of size 5^2 (Khuri-Makdisi large model)
```

```
>>> from sage.all import *
>>> # long time
>>> k = GF(Integer(5))
>>> P2 = ProjectiveSpace(k, Integer(2), names=('x', 'y', 'z')); (x, y, z,) = P2._
↪first_ngens(3)
>>> C = Curve(x**Integer(3) + z**Integer(3) - y**Integer(2)*z, P2)
>>> h = C.function(y/x).divisor_of_poles()
>>> J = C.jacobian(model='km_large', base_div=h)
>>> G1 = J.group()
>>> K = k.extension(Integer(2))
>>> G2 = J.group(K)
>>> G2.coerce_map_from(G1)
Jacobian group embedding map:
  From: Group of rational points of Jacobian
        over Finite Field of size 5 (Khuri-Makdisi large model)
  To:   Group of rational points of Jacobian
        over Finite Field in z2 of size 5^2 (Khuri-Makdisi large model)
```

```
class sage.rings.function_field.jacobian_khuri_makdisi.JacobianGroup_finite_field(parent,
                                                                                   func-
                                                                                   tion_field,
                                                                                   base_div)
```

Bases: *JacobianGroup*, *JacobianGroup_finite_field_base*

Jacobian groups of function fields over finite fields.

INPUT:

- *parent* – a Jacobian
- *function_field* – a function field
- *base_div* – an effective divisor of the function field

EXAMPLES:

```

sage: # long time
sage: k = GF(7)
sage: P2.<x,y,z> = ProjectiveSpace(k, 2)
sage: C = Curve(x^3 + 5*z^3 - y^2*z, P2)
sage: h = C.function(y/x).divisor_of_poles()
sage: J = C.jacobian(model='km_large', base_div=h)
sage: G1 = J.group()
sage: K = k.extension(2)
sage: G2 = J.group(K)
sage: G2.coerce_map_from(G1)
Jacobian group embedding map:
  From: Group of rational points of Jacobian
        over Finite Field of size 7 (Khuri-Makdisi large model)
  To:   Group of rational points of Jacobian
        over Finite Field in z2 of size 7^2 (Khuri-Makdisi large model)

```

```

>>> from sage.all import *
>>> # long time
>>> k = GF(Integer(7))
>>> P2 = ProjectiveSpace(k, Integer(2), names=('x', 'y', 'z')); (x, y, z,) = P2._
->first_ngens(3)
>>> C = Curve(x**Integer(3) + Integer(5)*z**Integer(3) - y**Integer(2)*z, P2)
>>> h = C.function(y/x).divisor_of_poles()
>>> J = C.jacobian(model='km_large', base_div=h)
>>> G1 = J.group()
>>> K = k.extension(Integer(2))
>>> G2 = J.group(K)
>>> G2.coerce_map_from(G1)
Jacobian group embedding map:
  From: Group of rational points of Jacobian
        over Finite Field of size 7 (Khuri-Makdisi large model)
  To:   Group of rational points of Jacobian
        over Finite Field in z2 of size 7^2 (Khuri-Makdisi large model)

```

Element

alias of *JacobianPoint_finite_field*

class sage.rings.function_field.jacobian_khuri_makdisi.**JacobianPoint** (*parent, w*)

Bases: *JacobianPoint_base*

Points of a Jacobian group.

INPUT:

- *parent* – Jacobian group
- *w* – matrix

EXAMPLES:

```

sage: # long time
sage: P2.<x,y,z> = ProjectiveSpace(GF(7), 2)
sage: C = Curve(x^3 + 5*z^3 - y^2*z, P2)
sage: b = C([0,1,0]).place()
sage: h = C.function(y/x).divisor_of_poles()

```

(continues on next page)

(continued from previous page)

```

sage: J = C.jacobian(model='km_large', base_div=h)
sage: G = J.group()
sage: p1 = C([3,2,1]).place()
sage: G.point(p1 - b)
Point of Jacobian determined by
[1 0 0 0 0 0 0 2 3]
[0 1 0 0 0 0 0 0 3]
[0 0 1 0 0 0 0 0 1]
[0 0 0 1 0 0 0 0 5]
[0 0 0 0 0 1 0 0 5]
[0 0 0 0 0 0 1 0 4]

```

```

>>> from sage.all import *
>>> # long time
>>> P2 = ProjectiveSpace(GF(Integer(7)), Integer(2), names=('x', 'y', 'z',)); (x, y, z) = P2._first_ngens(3)
>>> C = Curve(x**Integer(3) + Integer(5)*z**Integer(3) - y**Integer(2)*z, P2)
>>> b = C([Integer(0), Integer(1), Integer(0)]).place()
>>> h = C.function(y/x).divisor_of_poles()
>>> J = C.jacobian(model='km_large', base_div=h)
>>> G = J.group()
>>> p1 = C([Integer(3), Integer(2), Integer(1)]).place()
>>> G.point(p1 - b)
Point of Jacobian determined by
[1 0 0 0 0 0 0 2 3]
[0 1 0 0 0 0 0 0 3]
[0 0 1 0 0 0 0 0 1]
[0 0 0 1 0 0 0 0 5]
[0 0 0 0 0 1 0 0 5]
[0 0 0 0 0 0 1 0 4]

```

addflip (*other*)

Return the addflip of this and *other* point.

The addflip of two points is by definition the negative of the sum of the points. This operation is faster than addition in Jacobian arithmetic by Khuri-Makdisi algorithms.

EXAMPLES:

```

sage: # long time
sage: P2.<x,y,z> = ProjectiveSpace(GF(7), 2)
sage: C = Curve(x^3 + 5*z^3 - y^2*z, P2)
sage: h = C.function(y/x).divisor_of_poles()
sage: b = C([0,1,0]).place()
sage: p11 = C([-1,2,1]).place()
sage: p12 = C([3,2,1]).place()
sage: J = C.jacobian(model='km_large', base_div=h)
sage: G = J.group()
sage: p1 = G.point(p11 - b)
sage: p2 = G.point(p12 - b)
sage: p1.addflip(p2)
Point of Jacobian determined by
[1 0 0 0 0 0 0 2 6]

```

(continues on next page)

(continued from previous page)

```
[0 1 0 0 0 0 0 0 3]
[0 0 1 0 0 0 0 0 4]
[0 0 0 1 0 0 0 0 3]
[0 0 0 0 0 1 0 0 5]
[0 0 0 0 0 0 1 0 2]
sage: _ == -(p1 + p2)
True
```

```
>>> from sage.all import *
>>> # long time
>>> P2 = ProjectiveSpace(GF(Integer(7)), Integer(2), names=('x', 'y', 'z',));
↳ (x, y, z,) = P2._first_ngens(3)
>>> C = Curve(x**Integer(3) + Integer(5)*z**Integer(3) - y**Integer(2)*z, P2)
>>> h = C.function(y/x).divisor_of_poles()
>>> b = C([Integer(0), Integer(1), Integer(0)]).place()
>>> p11 = C([-Integer(1), Integer(2), Integer(1)]).place()
>>> p12 = C([Integer(3), Integer(2), Integer(1)]).place()
>>> J = C.jacobian(model='km_large', base_div=h)
>>> G = J.group()
>>> p1 = G.point(p11 - b)
>>> p2 = G.point(p12 - b)
>>> p1.addflip(p2)
Point of Jacobian determined by
[1 0 0 0 0 0 0 2 6]
[0 1 0 0 0 0 0 0 3]
[0 0 1 0 0 0 0 0 4]
[0 0 0 1 0 0 0 0 3]
[0 0 0 0 0 1 0 0 5]
[0 0 0 0 0 0 1 0 2]
>>> _ == -(p1 + p2)
True
```

defining_matrix()

Return the matrix whose row span determines the effective divisor representing this point.

EXAMPLES:

```
sage: # long time
sage: P2.<x,y,z> = ProjectiveSpace(GF(7), 2)
sage: C = Curve(x^3 + 5*z^3 - y^2*z, P2)
sage: h = C.function(y/x).divisor_of_poles()
sage: J = C.jacobian(model='km_large', base_div=h)
sage: G = J.group()
sage: b = C([0,1,0]).place()
sage: p1 = C([-1,2,1]).place()
sage: p = G.point(p1 - b)
sage: p.defining_matrix()
[1 0 0 0 0 0 0 2 5]
[0 1 0 0 0 0 0 0 3]
[0 0 1 0 0 0 0 0 2]
[0 0 0 1 0 0 0 0 6]
[0 0 0 0 0 1 0 0 5]
```

(continues on next page)

(continued from previous page)

```
[0 0 0 0 0 0 1 0 1]
```

```
>>> from sage.all import *
>>> # long time
>>> P2 = ProjectiveSpace(GF(Integer(7)), Integer(2), names=('x', 'y', 'z'));
↪ (x, y, z,) = P2._first_ngens(3)
>>> C = Curve(x**Integer(3) + Integer(5)*z**Integer(3) - y**Integer(2)*z, P2)
>>> h = C.function(y/x).divisor_of_poles()
>>> J = C.jacobian(model='km_large', base_div=h)
>>> G = J.group()
>>> b = C([Integer(0), Integer(1), Integer(0)]).place()
>>> p1 = C([-Integer(1), Integer(2), Integer(1)]).place()
>>> p = G.point(p1 - b)
>>> p.defined_matrix()
[1 0 0 0 0 0 0 2 5]
[0 1 0 0 0 0 0 0 3]
[0 0 1 0 0 0 0 0 2]
[0 0 0 1 0 0 0 0 6]
[0 0 0 0 0 1 0 0 5]
[0 0 0 0 0 0 1 0 1]
```

divisor()

Return the divisor representing this point.

EXAMPLES:

```
sage: # long time
sage: P2.<x,y,z> = ProjectiveSpace(GF(7), 2)
sage: C = Curve(x^3 + 5*z^3 - y^2*z, P2)
sage: F = C.function_field()
sage: h = C.function(y/x).divisor_of_poles()
sage: J = C.jacobian(model='km_large', base_div=h)
sage: G = J.group()
sage: b = F.get_place(1)
sage: p = C([-1, 2, 1]).place()
sage: pt = G.point(p - b)
sage: G.point(pt.divisor()) == pt
True
```

```
>>> from sage.all import *
>>> # long time
>>> P2 = ProjectiveSpace(GF(Integer(7)), Integer(2), names=('x', 'y', 'z'));
↪ (x, y, z,) = P2._first_ngens(3)
>>> C = Curve(x**Integer(3) + Integer(5)*z**Integer(3) - y**Integer(2)*z, P2)
>>> F = C.function_field()
>>> h = C.function(y/x).divisor_of_poles()
>>> J = C.jacobian(model='km_large', base_div=h)
>>> G = J.group()
>>> b = F.get_place(Integer(1))
>>> p = C([-Integer(1), Integer(2), Integer(1)]).place()
>>> pt = G.point(p - b)
>>> G.point(pt.divisor()) == pt
```

(continues on next page)

(continued from previous page)

True

ALGORITHM: Lemma 2.1 of [Khu2004].

multiple(*n*)Return the *n*-th multiple of this point.

INPUT:

- *n* – integer

EXAMPLES:

```
sage: # long time
sage: P2.<x,y,z> = ProjectiveSpace(GF(7), 2)
sage: C = Curve(x^3 + 5*z^3 - y^2*z, P2)
sage: h = C.function(y/x).divisor_of_poles()
sage: b = C([0,1,0]).place()
sage: pl = C([-1,2,1]).place()
sage: J = C.jacobian(model='km_large', base_div=h)
sage: G = J.group()
sage: p = G.point(pl - b)
sage: p.multiple(100)
Point of Jacobian determined by
[1 0 0 0 0 2 0 1 1]
[0 1 0 0 0 5 0 1 6]
[0 0 1 0 0 2 0 6 3]
[0 0 0 1 0 1 0 0 0]
[0 0 0 0 1 5 0 1 4]
[0 0 0 0 0 0 1 1 0]
```

```
>>> from sage.all import *
>>> # long time
>>> P2 = ProjectiveSpace(GF(Integer(7)), Integer(2), names=('x', 'y', 'z'));
↪ (x, y, z,) = P2._first_ngens(3)
>>> C = Curve(x**Integer(3) + Integer(5)*z**Integer(3) - y**Integer(2)*z, P2)
>>> h = C.function(y/x).divisor_of_poles()
>>> b = C([Integer(0), Integer(1), Integer(0)]).place()
>>> pl = C([-Integer(1), Integer(2), Integer(1)]).place()
>>> J = C.jacobian(model='km_large', base_div=h)
>>> G = J.group()
>>> p = G.point(pl - b)
>>> p.multiple(Integer(100))
Point of Jacobian determined by
[1 0 0 0 0 2 0 1 1]
[0 1 0 0 0 5 0 1 6]
[0 0 1 0 0 2 0 6 3]
[0 0 0 1 0 1 0 0 0]
[0 0 0 0 1 5 0 1 4]
[0 0 0 0 0 0 1 1 0]
```

```
class sage.rings.function_field.jacobian_khuri_makdisi.JacobianPoint_finite_field(parent,
                                                                                   w)
```

Bases: `JacobianPoint`, `JacobianPoint_finite_field_base`

26.4 Khuri-Makdisi algorithms for arithmetic in Jacobians

This module implements Khuri-Makdisi's algorithms of [Khu2004].

In the implementation, we use notations close to the ones used by Khuri-Makdisi. We describe them below for readers of the code.

Let D_0 be the base divisor of the Jacobian in Khuri-Makdisi model. So D_0 is an effective divisor of appropriate degree d_0 depending on the model. Let g be the genus of the underlying function field. For large and medium models, $d_0 \geq 2g + 1$. For small model $d_0 \geq g + 1$. A point of the Jacobian is a divisor class containing a divisor $D - D_0$ of degree 0 with an effective divisor D of degree d_0 .

Let V_n denote the vector space $H^0(O(nD_0))$ with a chosen basis, and let $\mu_{n,m}$ is a bilinear map from $V_n \times V_m \rightarrow V_{n+m}$ defined by $(f, g) \mapsto fg$. The map $\mu_{n,m}$ can be represented by a 3-dimensional array as depicted below:

```

      f
    *-----*
d / | e    / |
 * - | ---- * |
 | * ---- | - *
 | /      | /
 * -----*
```

where $d = \dim V_n$, $e = \dim V_m$, $f = \dim V_{n+m}$. In the implementation, we instead use a matrix of size $d \times ef$. Each row of the matrix denotes a matrix of size $e \times f$.

A point of the Jacobian is represented by an effective divisor D . In Khuri-Makdisi algorithms, the divisor D is represented by a subspace $W_D = H^0(O(n_0 D_0 - D))$ of V_{n_0} with fixed n_0 depending on the model. For large and small models, $n_0 = 3$ and $L = O(3D_0)$, and for medium model, $n_0 = 2$ and $L = O(2D_0)$.

The subspace W_D is the row space of a matrix w_D . Thus in the implementation, the matrix w_D represents a point of the Jacobian. The row space of the matrix w_L is $V_{n_0} = H^0(O(n_0 D_0))$.

The function `mu_image(w_D, w_E, mu_mat_n_m, expected_dim)` computes the image $\mu_{n,m}(W_D, W_E)$ of the expected dimension.

The function `mu_preimage(w_E, w_F, mu_mat_n_m, expected_codim)` computes the preimage W_D such that $\mu_{n,m}(W_D, W_E) = W_F$ of the expected codimension $\dim V_n - \dim W_D$, which is a multiple of d_0 .

AUTHORS:

- Kwankyu Lee (2022-01): initial version

```
class sage.rings.function_field.khuri_makdisi.KhuriMakdisi_base
```

Bases: object

add(wd1, wd2)

Theorem 4.5 (addition).

multiple(wd, n)

Compute multiple by additive square-and-multiply method.

negate(wd)

Theorem 4.4 (negation), first method.

subtract (*wd1*, *wd2*)

Theorem 4.6 (subtraction), first method.

zero_divisor ()

Return the matrix w_L representing zero divisor.

class sage.rings.function_field.khuri_makdisi.**KhuriMakdisi_large**

Bases: *KhuriMakdisi_base*

Khuri-Makdisi's large model.

add_divisor (*wd1*, *wd2*, *d1*, *d2*)

Theorem 3.6 (addition of divisors, first method).

We assume that w_{D_1} , w_{D_2} represent divisors of degree at most $3d_0 - 2g - 1$.

addflip (*wd1*, *wd2*)

Theorem 4.3 (addflip).

equal (*wd*, *we*)

Theorem 4.1, second method.

class sage.rings.function_field.khuri_makdisi.**KhuriMakdisi_medium**

Bases: *KhuriMakdisi_base*

Khuri-Makdisi's *medium* model

add_divisor (*wd1*, *wd2*, *d1*, *d2*)

Theorem 3.6 (addition of divisors, first method).

We assume that w_{D_1} , w_{D_2} represent divisors of degree at most $4d_0 - 2g - 1$.

addflip (*wd1*, *wd2*)

Theorem 5.1 (addflip in medium model).

equal (*wd*, *we*)

Theorem 4.1, second method.

class sage.rings.function_field.khuri_makdisi.**KhuriMakdisi_small**

Bases: *KhuriMakdisi_base*

Khuri-Makdisi's *small* model

add_divisor (*wd1*, *wd2*, *d1*, *d2*)

Theorem 3.6 (addition of divisors, first method).

We assume that w_{D_1} , w_{D_2} represent divisors of degree at most $6d_0 - 2g - 1$.

addflip (*wd1*, *wd2*)

Theorem 5.3 (addflip in small model), second method.

equal (*wd*, *we*)

Theorem 4.1, second method.

negate (*wd*)

Theorem 5.4 (negation in small model).

A SUPPORT MODULE

27.1 Hermite form computation for function fields

This module provides an optimized implementation of the algorithm computing Hermite forms of matrices over polynomials. This is the workhorse of the function field machinery of Sage.

EXAMPLES:

```
sage: P.<x> = PolynomialRing(QQ)
sage: A = matrix(P,3,[-(x-1)^((i-j+1) % 3) for i in range(3) for j in range(3)])
sage: A
[
      -x + 1          -1 -x^2 + 2*x - 1]
[-x^2 + 2*x - 1      -x + 1          -1]
[
      -1 -x^2 + 2*x - 1      -x + 1]
sage: from sage.rings.function_field.hermite_form_polynomial import reversed_hermite_
      ↪form
sage: B = copy(A)
sage: U = reversed_hermite_form(B, transformation=True)
sage: U * A == B
True
sage: B
[x^3 - 3*x^2 + 3*x - 2          0          0]
[
      0 x^3 - 3*x^2 + 3*x - 2          0]
[
      x^2 - 2*x + 1          x - 1          1]
```

```
>>> from sage.all import *
>>> P = PolynomialRing(QQ, names=('x',)); (x,) = P._first_ngens(1)
>>> A = matrix(P,Integer(3),[-(x-Integer(1))**((i-j+Integer(1)) % Integer(3)) for i_
      ↪in range(Integer(3)) for j in range(Integer(3))])
>>> A
[
      -x + 1          -1 -x^2 + 2*x - 1]
[-x^2 + 2*x - 1      -x + 1          -1]
[
      -1 -x^2 + 2*x - 1      -x + 1]
>>> from sage.rings.function_field.hermite_form_polynomial import reversed_hermite_
      ↪form
>>> B = copy(A)
>>> U = reversed_hermite_form(B, transformation=True)
>>> U * A == B
True
>>> B
[x^3 - 3*x^2 + 3*x - 2          0          0]
```

(continues on next page)

(continued from previous page)

```
[
      0 x^3 - 3*x^2 + 3*x - 2      0]
[
      x^2 - 2*x + 1      x - 1      1]
```

The function `reversed_hermite_form()` computes the reversed hermite form, which is reversed both row-wise and column-wise from the usual hermite form. Let us check it:

```
sage: A.reverse_rows_and_columns()
sage: C = copy(A.hermite_form())
sage: C.reverse_rows_and_columns()
sage: C
[x^3 - 3*x^2 + 3*x - 2      0      0]
[
      0 x^3 - 3*x^2 + 3*x - 2      0]
[
      x^2 - 2*x + 1      x - 1      1]
sage: C == B
True
```

```
>>> from sage.all import *
>>> A.reverse_rows_and_columns()
>>> C = copy(A.hermite_form())
>>> C.reverse_rows_and_columns()
>>> C
[x^3 - 3*x^2 + 3*x - 2      0      0]
[
      0 x^3 - 3*x^2 + 3*x - 2      0]
[
      x^2 - 2*x + 1      x - 1      1]
>>> C == B
True
```

AUTHORS:

- Kwankyu Lee (2021-05-21): initial version

`sage.rings.function_field.hermite_form_polynomial.reversed_hermite_form(mat, transformation=False)`

Transform the matrix in place to reversed hermite normal form and optionally return the transformation matrix.

INPUT:

- `transformation` – boolean (default: `False`); if `True`, return the transformation matrix

EXAMPLES:

```
sage: from sage.rings.function_field.hermite_form_polynomial import reversed_
      ↪hermite_form
sage: P.<x> = PolynomialRing(QQ)
sage: A = matrix(P, 3, [-(x-1)^(i-2*j) % 4] for i in range(3) for j in range(3))
sage: A
[
      -1      -x^2 + 2*x - 1      -1]
[
      -x + 1 -x^3 + 3*x^2 - 3*x + 1      -x + 1]
[
      -x^2 + 2*x - 1      -1      -x^2 + 2*x - 1]
sage: B = copy(A)
sage: U = reversed_hermite_form(B, transformation=True)
sage: U * A == B
True
sage: B
```

(continues on next page)

(continued from previous page)

```
[
          0          0          0]
[
    0 x^4 - 4*x^3 + 6*x^2 - 4*x    0]
[
          1          x^2 - 2*x + 1    1]
```

```
>>> from sage.all import *
>>> from sage.rings.function_field.hermite_form_polynomial import reversed_
    ↪ hermite_form
>>> P = PolynomialRing(QQ, names=('x',)); (x,) = P._first_ngens(1)
>>> A = matrix(P, Integer(3), [-(x-Integer(1))**((i-Integer(2)*j) % Integer(4)) for
    ↪ i in range(Integer(3)) for j in range(Integer(3))])
>>> A
[
          -1          -x^2 + 2*x - 1          -1]
[
          -x + 1 -x^3 + 3*x^2 - 3*x + 1          -x + 1]
[
    -x^2 + 2*x - 1          -1          -x^2 + 2*x - 1]
>>> B = copy(A)
>>> U = reversed_hermite_form(B, transformation=True)
>>> U * A == B
True
>>> B
[
          0          0          0]
[
    0 x^4 - 4*x^3 + 6*x^2 - 4*x    0]
[
          1          x^2 - 2*x + 1    1]
```


INDICES AND TABLES

- [Index](#)
- [Module Index](#)
- [Search Page](#)

PYTHON MODULE INDEX

r

sage.rings.function_field.constructor, 311
sage.rings.function_field.derivations, 283
sage.rings.function_field.derivations_polymod, 287
sage.rings.function_field.derivations_rational, 285
sage.rings.function_field.differential, 265
sage.rings.function_field.divisor, 253
sage.rings.function_field.element, 105
sage.rings.function_field.element_polymod, 135
sage.rings.function_field.element_rational, 127
sage.rings.function_field.extensions, 305
sage.rings.function_field.function_field, 3
sage.rings.function_field.function_field_polymod, 59
sage.rings.function_field.function_field_rational, 41
sage.rings.function_field.hermite_form_polynomial, 351
sage.rings.function_field.ideal, 183
sage.rings.function_field.ideal_polymod, 213
sage.rings.function_field.ideal_rational, 205
sage.rings.function_field.jacobian_base, 315
sage.rings.function_field.jacobian_hess, 327
sage.rings.function_field.jacobian_khuri_makdisi, 336
sage.rings.function_field.khuri_makdisi, 348
sage.rings.function_field.maps, 293
sage.rings.function_field.order, 139
sage.rings.function_field.order_basis, 151
sage.rings.function_field.order_polymod, 165
sage.rings.function_field.order_rational, 145
sage.rings.function_field.place, 237
sage.rings.function_field.place_polymod, 247
sage.rings.function_field.place_rational, 243
sage.rings.function_field.valuation_ring, 279

A

`add()` (*sage.rings.function_field.khuri_makdisi.KhuriMakdisi_base* method), 348

`add_divisor()` (*sage.rings.function_field.khuri_makdisi.KhuriMakdisi_large* method), 349

`add_divisor()` (*sage.rings.function_field.khuri_makdisi.KhuriMakdisi_medium* method), 349

`add_divisor()` (*sage.rings.function_field.khuri_makdisi.KhuriMakdisi_small* method), 349

`addflip()` (*sage.rings.function_field.jacobian_hess.JacobianPoint* method), 333

`addflip()` (*sage.rings.function_field.jacobian_khuri_makdisi.JacobianPoint* method), 344

`addflip()` (*sage.rings.function_field.khuri_makdisi.KhuriMakdisi_large* method), 349

`addflip()` (*sage.rings.function_field.khuri_makdisi.KhuriMakdisi_medium* method), 349

`addflip()` (*sage.rings.function_field.khuri_makdisi.KhuriMakdisi_small* method), 349

B

`base_curve()` (*sage.rings.function_field.jacobian_base.Jacobian_base* method), 324

`base_divisor()` (*sage.rings.function_field.jacobian_base.Jacobian_base* method), 325

`base_divisor()` (*sage.rings.function_field.jacobian_base.JacobianGroup_base* method), 319

`base_field()` (*sage.rings.function_field.function_field_polymod.FunctionField_polymod* method), 71

`base_field()` (*sage.rings.function_field.function_field_rational.RationalFunctionField* method), 44

`base_ring()` (*sage.rings.function_field.ideal.FunctionFieldIdeal* method), 187

`basis()` (*sage.rings.function_field.differential.DifferentialSpace* method), 267

`basis()` (*sage.rings.function_field.order_basis.FunctionFieldOrder_basis* method), 158

`basis()` (*sage.rings.function_field.order_basis.FunctionFieldOrderInfinite_basis* method), 152

`basis()` (*sage.rings.function_field.order_polymod.FunctionFieldMaximalOrder_polymod* method), 174

`basis()` (*sage.rings.function_field.order_polymod.FunctionFieldMaximalOrderInfinite_polymod* method), 166

`basis()` (*sage.rings.function_field.order_rational.FunctionFieldMaximalOrder_rational* method), 147

`basis()` (*sage.rings.function_field.order_rational.FunctionFieldMaximalOrderInfinite_rational* method), 145

`basis_differential_space()` (*sage.rings.function_field.divisor.FunctionFieldDivisor* method), 255

`basis_function_space()` (*sage.rings.function_field.divisor.FunctionFieldDivisor* method), 256

`basis_matrix()` (*sage.rings.function_field.ideal_polymod.FunctionFieldIdeal_polymod* method), 222

`basis_of_differentials_of_first_kind()` (*sage.rings.function_field.function_field.FunctionField* method), 10

`basis_of_holomorphic_differentials()` (*sage.rings.function_field.function_field.FunctionField* method), 11

C

`cartier()` (*sage.rings.function_field.differential.FunctionFieldDifferential_global* method), 276

`change_variable_name()` (*sage.rings.function_field.function_field_polymod.FunctionField_polymod* method), 71

`change_variable_name()` (*sage.rings.function_field.function_field_rational.RationalFunctionField* method), 44

`characteristic()` (*sage.rings.function_field.function_field.FunctionField* method), 12

`characteristic_polynomial()` (*sage.rings.function_field.element.FunctionFieldElement* method), 107

`charpoly()` (*sage.rings.function_field.element.Function-*

FieldElement method), 107

`codifferent()` (*sage.rings.function_field.order_polymod.FunctionFieldMaximalOrder_polymod* method), 174

`codomain()` (*sage.rings.function_field.maps.MapVectorSpaceToFunctionField* method), 302

`completion()` (*sage.rings.function_field.function_field.FunctionField* method), 12

`conorm_divisor()` (*sage.rings.function_field.extensions.ConstantFieldExtension* method), 307

`conorm_place()` (*sage.rings.function_field.extensions.ConstantFieldExtension* method), 307

`constant_base_field()` (*sage.rings.function_field.function_field_polymod.FunctionField_polymod* method), 73

`constant_base_field()` (*sage.rings.function_field.function_field_rational.RationalFunctionField* method), 45

`constant_field()` (*sage.rings.function_field.function_field_polymod.FunctionField_polymod* method), 74

`constant_field()` (*sage.rings.function_field.function_field_polymod.FunctionField_simple* method), 98

`constant_field()` (*sage.rings.function_field.function_field_rational.RationalFunctionField* method), 45

`ConstantFieldExtension` (class in *sage.rings.function_field.extensions*), 307

`construction()` (*sage.rings.function_field.jacobian_base.JacobianGroup_base* method), 319

`coordinate_vector()` (*sage.rings.function_field.order_basis.FunctionFieldOrder_basis* method), 158

`coordinate_vector()` (*sage.rings.function_field.order_polymod.FunctionFieldMaximalOrder_polymod* method), 175

`coordinate_vector()` (*sage.rings.function_field.order_polymod.FunctionFieldMaximalOrderInfinite_polymod* method), 166

`create_key()` (*sage.rings.function_field.constructor.FunctionFieldExtensionFactory* method), 312

`create_key()` (*sage.rings.function_field.constructor.FunctionFieldFactory* method), 313

`create_object()` (*sage.rings.function_field.constructor.FunctionFieldExtensionFactory* method), 312

`create_object()` (*sage.rings.function_field.constructor.FunctionFieldFactory* method), 314

`curve()` (*sage.rings.function_field.jacobian_base.Jacobian_base* method), 325

D

`decomposition()` (*sage.rings.function_field.order_polymod.FunctionFieldMaximalOrder_global* method), 172

`decomposition()` (*sage.rings.function_field.order_polymod.FunctionFieldMaximalOrder_polymod* method), 176

`decomposition()` (*sage.rings.function_field.order_polymod.FunctionFieldMaximalOrderInfinite_polymod* method), 167

`default_precision()` (*sage.rings.function_field.maps.FunctionFieldCompletion* method), 296

`defining_divisor()` (*sage.rings.function_field.jacobian_hess.JacobianPoint* method), 333

`defining_matrix()` (*sage.rings.function_field.jacobian_khuri_makdisi.JacobianPoint* method), 345

`defining_morphism()` (*sage.rings.function_field.extensions.ConstantFieldExtension* method), 308

`degree()` (*sage.rings.function_field.divisor.FunctionFieldDivisor* method), 256

`degree()` (*sage.rings.function_field.element.FunctionFieldElement* method), 108

`degree()` (*sage.rings.function_field.function_field_polymod.FunctionField_polymod* method), 74

`degree()` (*sage.rings.function_field.function_field_rational.RationalFunctionField* method), 46

`degree()` (*sage.rings.function_field.place_polymod.FunctionFieldPlace_polymod* method), 247

`degree()` (*sage.rings.function_field.place_rational.FunctionFieldPlace_rational* method), 243

`denominator()` (*sage.rings.function_field.divisor.FunctionFieldDivisor* method), 257

`denominator()` (*sage.rings.function_field.element_rational.FunctionFieldElement_rational* method), 127

`denominator()` (*sage.rings.function_field.ideal_polymod.FunctionFieldIdeal_polymod* method), 223

`denominator()` (*sage.rings.function_field.ideal_rational.FunctionFieldIdeal_rational* method), 208

`derivative()` (*sage.rings.function_field.element.FunctionFieldElement* method), 108

`dict()` (*sage.rings.function_field.divisor.FunctionFieldDivisor* method), 257

`different()` (*sage.rings.function_field.function_field_polymod.FunctionField_polymod* method), 75

`different()` (*sage.rings.function_field.function_field_rational.RationalFunctionField* method), 46

`different()` (*sage.rings.function_field.order_polymod.FunctionFieldMaximalOrder_polymod*

- method), 177
- `different()` (*sage.rings.function_field.order_polymod.FunctionFieldMaximalOrderInfinite_polymod* method), 169
- `differential()` (*sage.rings.function_field.element.FunctionFieldElement* method), 109
- `differential_space()` (*sage.rings.function_field.divisor.FunctionFieldDivisor* method), 258
- `DifferentialsSpace` (class in *sage.rings.function_field.differential*), 266
- `DifferentialsSpace_global` (class in *sage.rings.function_field.differential*), 270
- `DifferentialsSpaceInclusion` (class in *sage.rings.function_field.differential*), 268
- `dimension()` (*sage.rings.function_field.divisor.FunctionFieldDivisor* method), 258
- `divisor()` (in module *sage.rings.function_field.divisor*), 262
- `divisor()` (*sage.rings.function_field.differential.FunctionFieldDifferential* method), 271
- `divisor()` (*sage.rings.function_field.element.FunctionFieldElement* method), 110
- `divisor()` (*sage.rings.function_field.ideal.FunctionFieldIdeal* method), 187
- `divisor()` (*sage.rings.function_field.jacobian_hess.JacobianPoint* method), 334
- `divisor()` (*sage.rings.function_field.jacobian_khuri_makdisi.JacobianPoint* method), 346
- `divisor()` (*sage.rings.function_field.place.FunctionFieldPlace* method), 239
- `divisor_group()` (*sage.rings.function_field.function_field.FunctionField* method), 16
- `divisor_of_poles()` (*sage.rings.function_field.element.FunctionFieldElement* method), 111
- `divisor_of_poles()` (*sage.rings.function_field.ideal.FunctionFieldIdeal* method), 189
- `divisor_of_zeros()` (*sage.rings.function_field.element.FunctionFieldElement* method), 111
- `divisor_of_zeros()` (*sage.rings.function_field.ideal.FunctionFieldIdeal* method), 190
- `DivisorGroup` (class in *sage.rings.function_field.divisor*), 254
- `domain()` (*sage.rings.function_field.maps.MapVectorSpaceToFunctionField* method), 302
- E**
- `Element` (*sage.rings.function_field.differential.DifferentialsSpace* attribute), 267
- `Element` (*sage.rings.function_field.differential.DifferentialsSpace_global* attribute), 270
- `Element` (*sage.rings.function_field.divisor.DivisorGroup* attribute), 254
- `Element` (*sage.rings.function_field.function_field_polymod.FunctionField_polymod* attribute), 71
- `Element` (*sage.rings.function_field.function_field_rational.RationalFunctionField* attribute), 44
- `Element` (*sage.rings.function_field.jacobian_hess.JacobianGroup* attribute), 329
- `Element` (*sage.rings.function_field.jacobian_hess.JacobianGroup_finite_field* attribute), 332
- `Element` (*sage.rings.function_field.jacobian_khuri_makdisi.JacobianGroup* attribute), 340
- `Element` (*sage.rings.function_field.jacobian_khuri_makdisi.JacobianGroup_finite_field* attribute), 343
- `Element` (*sage.rings.function_field.place.PlaceSet* attribute), 241
- `element()` (*sage.rings.function_field.element_polymod.FunctionFieldElement_polymod* method), 135
- `element()` (*sage.rings.function_field.element_rational.FunctionFieldElement_rational* method), 127
- `equal()` (*sage.rings.function_field.khuri_makdisi.KhuriMakdisi_large* method), 349
- `equal()` (*sage.rings.function_field.khuri_makdisi.KhuriMakdisi_medium* method), 349
- `equal()` (*sage.rings.function_field.khuri_makdisi.KhuriMakdisi_small* method), 349
- `equation_order()` (*sage.rings.function_field.function_field_polymod.FunctionField_integral* method), 67
- `equation_order()` (*sage.rings.function_field.function_field_polymod.FunctionField_polymod* method), 76
- `equation_order()` (*sage.rings.function_field.function_field_rational.RationalFunctionField* method), 46
- `equation_order_infinite()` (*sage.rings.function_field.function_field_polymod.FunctionField_integral* method), 68
- `equation_order_infinite()` (*sage.rings.function_field.function_field_rational.RationalFunctionField* method), 47
- `evaluate()` (*sage.rings.function_field.element.FunctionFieldElement* method), 112
- `exact_constant_field()` (*sage.rings.function_field.function_field_polymod.FunctionField_simple* method), 99
- `extension()` (*sage.rings.function_field.function_field_rational.RationalFunctionField* method), 47
- `extension()` (*sage.rings.function_field.function_field.FunctionField* method), 17
- `extension_constant_field()` (*sage.rings.function*

tion_field.function_field.FunctionField method),
18

F

facade_for() (*sage.rings.function_field.jacobian_base.Jacobian_base* method), 325

factor() (*sage.rings.function_field.element_rational.FunctionFieldElement_rational* method),
128

factor() (*sage.rings.function_field.ideal.FunctionFieldIdeal* method), 191

field() (*sage.rings.function_field.function_field_rational.RationalFunctionField* method), 48

fraction_field() (*sage.rings.function_field.order.FunctionFieldOrder_base* method), 143

FractionFieldToFunctionField (class in *sage.rings.function_field.maps*), 294

free_module() (*sage.rings.function_field.function_field_polymod.FunctionField_polymod* method), 76

free_module() (*sage.rings.function_field.function_field_rational.RationalFunctionField* method), 48

free_module() (*sage.rings.function_field.order_basis.FunctionFieldOrder_basis* method), 159

free_module() (*sage.rings.function_field.order_basis.FunctionFieldOrderInfinite_basis* method),
153

free_module() (*sage.rings.function_field.order_polymod.FunctionFieldMaximalOrder_polymod* method), 177

frobenius() (*sage.rings.function_field.jacobian_base.JacobianPoint_finite_field_base* method), 323

function_field() (*sage.rings.function_field.differential.DifferentialsSpace* method), 268

function_field() (*sage.rings.function_field.divisor.DivisorGroup* method), 254

function_field() (*sage.rings.function_field.jacobian_base.JacobianGroup_base* method), 320

function_field() (*sage.rings.function_field.order.FunctionFieldOrder_base* method), 143

function_field() (*sage.rings.function_field.place.FunctionFieldPlace* method),
239

function_field() (*sage.rings.function_field.place.PlaceSet* method), 241

function_space() (*sage.rings.function_field.divisor.FunctionFieldDivisor* method), 259

FunctionField (class in *sage.rings.function_field.function_field*), 10

FunctionField_char_zero (class in *sage.rings.function_field.function_field_polymod*), 59

FunctionField_char_zero_integral (class in

sage.rings.function_field.function_field_polymod), 60

FunctionField_global (class in *sage.rings.function_field.function_field_polymod*), 60

FunctionField_global_integral (class in *sage.rings.function_field.function_field_polymod*), 67

FunctionField_integral (class in *sage.rings.function_field.function_field_polymod*), 67

FunctionField_polymod (class in *sage.rings.function_field.function_field_polymod*), 69

FunctionField_simple (class in *sage.rings.function_field.function_field_polymod*), 98

FunctionFieldCompletion (class in *sage.rings.function_field.maps*), 295

FunctionFieldConversionToConstantBaseField (class in *sage.rings.function_field.maps*), 297

FunctionFieldDerivation (class in *sage.rings.function_field.derivations*), 283

FunctionFieldDerivation_inseparable (class in *sage.rings.function_field.derivations_polymod*),
287

FunctionFieldDerivation_rational (class in *sage.rings.function_field.derivations_rational*),
285

FunctionFieldDerivation_separable (class in *sage.rings.function_field.derivations_polymod*),
288

FunctionFieldDifferential (class in *sage.rings.function_field.differential*), 270

FunctionFieldDifferential_global (class in *sage.rings.function_field.differential*), 275

FunctionFieldDivisor (class in *sage.rings.function_field.divisor*), 255

FunctionFieldElement (class in *sage.rings.function_field.element*), 106

FunctionFieldElement_polymod (class in *sage.rings.function_field.element_polymod*),
135

FunctionFieldElement_rational (class in *sage.rings.function_field.element_rational*),
127

FunctionFieldExtension (class in *sage.rings.function_field.extensions*), 309

FunctionFieldExtensionFactory (class in *sage.rings.function_field.constructor*), 311

FunctionFieldFactory (class in *sage.rings.function_field.constructor*), 313

FunctionFieldHigherDerivation (class in *sage.rings.function_field.derivations_polymod*),
288

FunctionFieldHigherDerivation_char_zero (class in *sage.rings.function_field.derivations_polymod*), 289

- FunctionFieldHigherDerivation_global (class in *sage.rings.function_field.derivations_polymod*), 290
- FunctionFieldIdeal (class in *sage.rings.function_field.ideal*), 186
- FunctionFieldIdeal_global (class in *sage.rings.function_field.ideal_polymod*), 219
- FunctionFieldIdeal_module (class in *sage.rings.function_field.ideal*), 198
- FunctionFieldIdeal_polymod (class in *sage.rings.function_field.ideal_polymod*), 221
- FunctionFieldIdeal_rational (class in *sage.rings.function_field.ideal_rational*), 207
- FunctionFieldIdealInfinite (class in *sage.rings.function_field.ideal*), 197
- FunctionFieldIdealInfinite_module (class in *sage.rings.function_field.ideal*), 197
- FunctionFieldIdealInfinite_polymod (class in *sage.rings.function_field.ideal_polymod*), 213
- FunctionFieldIdealInfinite_rational (class in *sage.rings.function_field.ideal_rational*), 205
- FunctionFieldLinearMap (class in *sage.rings.function_field.maps*), 297
- FunctionFieldLinearMapSection (class in *sage.rings.function_field.maps*), 297
- FunctionFieldMaximalOrder (class in *sage.rings.function_field.order*), 142
- FunctionFieldMaximalOrder_global (class in *sage.rings.function_field.order_polymod*), 172
- FunctionFieldMaximalOrder_polymod (class in *sage.rings.function_field.order_polymod*), 173
- FunctionFieldMaximalOrder_rational (class in *sage.rings.function_field.order_rational*), 147
- FunctionFieldMaximalOrderInfinite (class in *sage.rings.function_field.order*), 142
- FunctionFieldMaximalOrderInfinite_polymod (class in *sage.rings.function_field.order_polymod*), 165
- FunctionFieldMaximalOrderInfinite_rational (class in *sage.rings.function_field.order_rational*), 145
- FunctionFieldMorphism (class in *sage.rings.function_field.maps*), 297
- FunctionFieldMorphism_polymod (class in *sage.rings.function_field.maps*), 298
- FunctionFieldMorphism_rational (class in *sage.rings.function_field.maps*), 298
- FunctionFieldOrder (class in *sage.rings.function_field.order*), 142
- FunctionFieldOrder_base (class in *sage.rings.function_field.order*), 142
- FunctionFieldOrder_basis (class in *sage.rings.function_field.order_basis*), 156
- FunctionFieldOrderInfinite (class in *sage.rings.function_field.order*), 142
- FunctionFieldOrderInfinite_basis (class in *sage.rings.function_field.order_basis*), 151
- FunctionFieldPlace (class in *sage.rings.function_field.place*), 238
- FunctionFieldPlace_polymod (class in *sage.rings.function_field.place_polymod*), 247
- FunctionFieldPlace_rational (class in *sage.rings.function_field.place_rational*), 243
- FunctionFieldRingMorphism (class in *sage.rings.function_field.maps*), 298
- FunctionFieldToFractionField (class in *sage.rings.function_field.maps*), 298
- FunctionFieldValuationRing (class in *sage.rings.function_field.valuation_ring*), 280
- FunctionFieldVectorSpaceIsomorphism (class in *sage.rings.function_field.maps*), 299
- ## G
- gaps () (*sage.rings.function_field.function_field_polymod.FunctionField_global* method), 62
- gaps () (*sage.rings.function_field.place_polymod.FunctionFieldPlace_polymod* method), 247
- gen () (*sage.rings.function_field.function_field_polymod.FunctionField_polymod* method), 80
- gen () (*sage.rings.function_field.function_field_rational.RationalFunctionField* method), 49
- gen () (*sage.rings.function_field.ideal_rational.FunctionFieldIdeal_rational* method), 208
- gen () (*sage.rings.function_field.ideal_rational.FunctionFieldIdealInfinite_rational* method), 205
- gen () (*sage.rings.function_field.ideal.FunctionFieldIdeal_module* method), 199
- gen () (*sage.rings.function_field.order_polymod.FunctionFieldMaximalOrder_polymod* method), 178
- gen () (*sage.rings.function_field.order_polymod.FunctionFieldMaximalOrderInfinite_polymod* method), 169
- gen () (*sage.rings.function_field.order_rational.FunctionFieldMaximalOrder_rational* method), 148
- gens () (*sage.rings.function_field.ideal_polymod.FunctionFieldIdeal_global* method), 220
- gens () (*sage.rings.function_field.ideal_polymod.FunctionFieldIdeal_polymod* method), 224
- gens () (*sage.rings.function_field.ideal_polymod.FunctionFieldIdealInfinite_polymod* method), 213
- gens () (*sage.rings.function_field.ideal_rational.FunctionFieldIdeal_rational* method), 208
- gens () (*sage.rings.function_field.ideal_rational.FunctionFieldIdealInfinite_rational* method), 206
- gens () (*sage.rings.function_field.ideal.FunctionFieldIdeal_module* method), 200

- gens_over_base() (sage.rings.function_field.ideal_polymod.FunctionFieldIdeal_polymod method), 224
- gens_over_base() (sage.rings.function_field.ideal_polymod.FunctionFieldIdealInfinite_polymod method), 214
- gens_over_base() (sage.rings.function_field.ideal_rational.FunctionFieldIdeal_rational method), 209
- gens_over_base() (sage.rings.function_field.ideal_rational.FunctionFieldIdealInfinite_rational method), 206
- gens_reduced() (sage.rings.function_field.ideal.FunctionFieldIdeal method), 193
- gens_two() (sage.rings.function_field.ideal_polymod.FunctionFieldIdeal_global method), 220
- gens_two() (sage.rings.function_field.ideal_polymod.FunctionFieldIdealInfinite_polymod method), 215
- genus() (sage.rings.function_field.function_field_polymod.FunctionField_polymod method), 81
- genus() (sage.rings.function_field.function_field_polymod.FunctionField_simple method), 100
- genus() (sage.rings.function_field.function_field_rational.RationalFunctionField method), 50
- get_place() (sage.rings.function_field.function_field_polymod.FunctionField_global method), 62
- get_place() (sage.rings.function_field.function_field_rational.RationalFunctionField_global method), 54
- get_points() (sage.rings.function_field.jacobian_base.JacobianGroup_finite_field_base method), 321
- group() (sage.rings.function_field.jacobian_base.Jacobian_base method), 326
- ## H
- higher_derivation() (sage.rings.function_field.function_field_polymod.FunctionField_char_zero method), 59
- higher_derivation() (sage.rings.function_field.function_field_polymod.FunctionField_global method), 63
- higher_derivation() (sage.rings.function_field.function_field_rational.RationalFunctionField_char_zero method), 54
- higher_derivation() (sage.rings.function_field.function_field_rational.RationalFunctionField_global method), 55
- higher_derivative() (sage.rings.function_field.element.FunctionFieldElement method), 113
- hilbert_symbol() (sage.rings.function_field.function_field.FunctionField method), 19
- hnf() (sage.rings.function_field.ideal_polymod.FunctionFieldIdeal_polymod method), 225
- hom() (sage.rings.function_field.function_field_polymod.FunctionField_polymod method), 81
- hom() (sage.rings.function_field.function_field_rational.RationalFunctionField method), 50
- ## I
- ideal() (sage.rings.function_field.order_basis.FunctionFieldOrder_basis method), 160
- ideal() (sage.rings.function_field.order_basis.FunctionFieldOrderInfinite_basis method), 153
- ideal() (sage.rings.function_field.order_polymod.FunctionFieldMaximalOrder_polymod method), 179
- ideal() (sage.rings.function_field.order_polymod.FunctionFieldMaximalOrderInfinite_polymod method), 170
- ideal() (sage.rings.function_field.order_rational.FunctionFieldMaximalOrder_rational method), 148
- ideal() (sage.rings.function_field.order_rational.FunctionFieldMaximalOrderInfinite_rational method), 146
- ideal_below() (sage.rings.function_field.ideal_polymod.FunctionFieldIdeal_polymod method), 226
- ideal_below() (sage.rings.function_field.ideal_polymod.FunctionFieldIdealInfinite_polymod method), 215
- ideal_monoid() (sage.rings.function_field.order.FunctionFieldOrder_base method), 143
- ideal_with_gens_over_base() (sage.rings.function_field.order_basis.FunctionFieldOrder_basis method), 161
- ideal_with_gens_over_base() (sage.rings.function_field.order_basis.FunctionFieldOrderInfinite_basis method), 154
- ideal_with_gens_over_base() (sage.rings.function_field.order_polymod.FunctionFieldMaximalOrder_polymod method), 180
- ideal_with_gens_over_base() (sage.rings.function_field.order_polymod.FunctionFieldMaximalOrderInfinite_polymod method), 171
- ideal_with_gens_over_base() (sage.rings.function_field.order_rational.FunctionFieldMaximalOrder_rational method), 149
- IdealMonoid (class in sage.rings.function_field.ideal), 202
- intersect() (sage.rings.function_field.ideal_polymod.FunctionFieldIdeal_polymod method), 228
- intersection() (sage.rings.function_field.ideal.FunctionFieldIdeal_module method), 200

- `inverse_mod()` (*sage.rings.function_field.element_rational.FunctionFieldElement_rational method*), 129
`is_effective()` (*sage.rings.function_field.divisor.FunctionFieldDivisor method*), 259
`is_field()` (*sage.rings.function_field.order.FunctionFieldOrder_base method*), 143
`is_finite()` (*sage.rings.function_field.function_field.FunctionField method*), 21
`is_FunctionField()` (in module *sage.rings.function_field.function_field*), 38
`is_FunctionFieldElement()` (in module *sage.rings.function_field.element*), 125
`is_global()` (*sage.rings.function_field.function_field.FunctionField method*), 21
`is_infinite_place()` (*sage.rings.function_field.place_polymod.FunctionFieldPlace_polymod method*), 248
`is_infinite_place()` (*sage.rings.function_field.place_rational.FunctionFieldPlace_rational method*), 243
`is_injective()` (*sage.rings.function_field.derivations.FunctionFieldDerivation method*), 284
`is_injective()` (*sage.rings.function_field.differential.DifferentialsSpaceInclusion method*), 269
`is_injective()` (*sage.rings.function_field.maps.FunctionFieldVectorSpaceIsomorphism method*), 300
`is_integral()` (*sage.rings.function_field.element.FunctionFieldElement method*), 114
`is_integral()` (*sage.rings.function_field.ideal_polymod.FunctionFieldIdeal_polymod method*), 228
`is_noetherian()` (*sage.rings.function_field.order.FunctionFieldOrder_base method*), 143
`is_nth_power()` (*sage.rings.function_field.element_polymod.FunctionFieldElement_polymod method*), 135
`is_nth_power()` (*sage.rings.function_field.element_rational.FunctionFieldElement_rational method*), 129
`is_nth_power()` (*sage.rings.function_field.element.FunctionFieldElement method*), 114
`is_perfect()` (*sage.rings.function_field.function_field.FunctionField method*), 22
`is_prime()` (*sage.rings.function_field.ideal_polymod.FunctionFieldIdeal_polymod method*), 230
`is_prime()` (*sage.rings.function_field.ideal_polymod.FunctionFieldIdealInfinite_polymod method*), 216
`is_prime()` (*sage.rings.function_field.ideal_rational.FunctionFieldIdeal_rational method*), 209
`is_prime()` (*sage.rings.function_field.ideal_rational.FunctionFieldIdealInfinite_rational method*), 206
`is_separable()` (*sage.rings.function_field.function_field_polymod.FunctionField_polymod method*), 84
`is_square()` (*sage.rings.function_field.element_rational.FunctionFieldElement_rational method*), 130
`is_subring()` (*sage.rings.function_field.order.FunctionFieldOrder_base method*), 144
`is_surjective()` (*sage.rings.function_field.differential.DifferentialsSpaceInclusion method*), 269
`is_surjective()` (*sage.rings.function_field.maps.FunctionFieldVectorSpaceIsomorphism method*), 300
- ## J
- `Jacobian` (class in *sage.rings.function_field.jacobian_hess*), 329
`Jacobian` (class in *sage.rings.function_field.jacobian_khuri_makdisi*), 339
`jacobian()` (*sage.rings.function_field.function_field.FunctionField method*), 22
`Jacobian_base` (class in *sage.rings.function_field.jacobian_base*), 324
`JacobianGroup` (class in *sage.rings.function_field.jacobian_hess*), 329
`JacobianGroup` (class in *sage.rings.function_field.jacobian_khuri_makdisi*), 339
`JacobianGroup_base` (class in *sage.rings.function_field.jacobian_base*), 318
`JacobianGroup_finite_field` (class in *sage.rings.function_field.jacobian_hess*), 331
`JacobianGroup_finite_field` (class in *sage.rings.function_field.jacobian_khuri_makdisi*), 342
`JacobianGroup_finite_field_base` (class in *sage.rings.function_field.jacobian_base*), 321
`JacobianGroupEmbedding` (class in *sage.rings.function_field.jacobian_hess*), 330
`JacobianGroupEmbedding` (class in *sage.rings.function_field.jacobian_khuri_makdisi*), 341
`JacobianGroupFunctor` (class in *sage.rings.function_field.jacobian_base*), 317
`JacobianPoint` (class in *sage.rings.function_field.jacobian_hess*), 332
`JacobianPoint` (class in *sage.rings.function_field.jacobian_khuri_makdisi*), 343
`JacobianPoint_base` (class in *sage.rings.function_field.jacobian_base*), 322
`JacobianPoint_finite_field` (class in *sage.rings.function_field.jacobian_hess*), 335
`JacobianPoint_finite_field` (class in *sage.rings.function_field.jacobian_khuri_makdisi*), 347

JacobianPoint_finite_field_base (class in *sage.rings.function_field.jacobian_base*), 322

K

KhuriMakdisi_base (class in *sage.rings.function_field.khuri_makdisi*), 348

KhuriMakdisi_large (class in *sage.rings.function_field.khuri_makdisi*), 349

KhuriMakdisi_medium (class in *sage.rings.function_field.khuri_makdisi*), 349

KhuriMakdisi_small (class in *sage.rings.function_field.khuri_makdisi*), 349

L

L_polynomial() (*sage.rings.function_field.function_field_polymod.FunctionField_global* method), 61

list() (*sage.rings.function_field.divisor.FunctionFieldDivisor* method), 260

list() (*sage.rings.function_field.element_polymod.FunctionFieldElement_polymod* method), 136

list() (*sage.rings.function_field.element_rational.FunctionFieldElement_rational* method), 131

local_uniformizer() (*sage.rings.function_field.place_polymod.FunctionFieldPlace_polymod* method), 249

local_uniformizer() (*sage.rings.function_field.place_rational.FunctionFieldPlace_rational* method), 243

M

make_FunctionFieldElement() (in module *sage.rings.function_field.element*), 126

MapFunctionFieldToVectorSpace (class in *sage.rings.function_field.maps*), 301

MapVectorSpaceToFunctionField (class in *sage.rings.function_field.maps*), 301

matrix() (*sage.rings.function_field.element.FunctionFieldElement* method), 115

maximal_order() (*sage.rings.function_field.function_field_polymod.FunctionField_global* method), 64

maximal_order() (*sage.rings.function_field.function_field_polymod.FunctionField_polymod* method), 86

maximal_order() (*sage.rings.function_field.function_field_rational.RationalFunctionField* method), 51

maximal_order_infinite() (*sage.rings.function_field.function_field_polymod.FunctionField_polymod* method), 86

maximal_order_infinite() (*sage.rings.function_field.function_field_rational.RationalFunctionField* method), 51

merge() (*sage.rings.function_field.jacobian_base.JacobianGroupFunctor* method), 317

minimal_polynomial() (*sage.rings.function_field.element.FunctionFieldElement* method), 118

minpoly() (*sage.rings.function_field.element.FunctionFieldElement* method), 118

module

sage.rings.function_field.constructor, 311

sage.rings.function_field.derivations, 283

sage.rings.function_field.derivations_polymod, 287

sage.rings.function_field.derivations_rational, 285

sage.rings.function_field.differential, 265

sage.rings.function_field.divisor, 253

sage.rings.function_field.element, 105

sage.rings.function_field.element_polymod, 135

sage.rings.function_field.element_rational, 127

sage.rings.function_field.extensions, 305

sage.rings.function_field.function_field, 3

sage.rings.function_field.function_field_polymod, 59

sage.rings.function_field.function_field_rational, 41

sage.rings.function_field.hermite_form_polynomial, 351

sage.rings.function_field.ideal, 183

sage.rings.function_field.ideal_polymod, 213

sage.rings.function_field.ideal_rational, 205

sage.rings.function_field.jacobian_base, 315

sage.rings.function_field.jacobian_hess, 327

sage.rings.function_field.jacobian_khuri_makdisi, 336

sage.rings.function_field.khuri_makdisi, 348

sage.rings.function_field.maps, 293

sage.rings.function_field.order, 139

sage.rings.function_field.order_basis, 151

sage.rings.function_field.order_polymod, 165

sage.rings.function_field.order_rational, 145

`sage.rings.function_field.place`, 237
`sage.rings.function_field.place_polymod`, 247
`sage.rings.function_field.place_rational`, 243
`sage.rings.function_field.valuation_ring`, 279
`module()` (`sage.rings.function_field.ideal_polymod.FunctionFieldIdeal_polymod` method), 231
`module()` (`sage.rings.function_field.ideal_rational.FunctionFieldIdeal_rational` method), 210
`module()` (`sage.rings.function_field.ideal.FunctionFieldIdeal_module` method), 201
`module()` (`sage.rings.function_field.ideal.FunctionFieldIdealInfinite_module` method), 197
`monic_integral_model()` (`sage.rings.function_field.function_field_polymod.FunctionField_polymod` method), 87
`monomial_coefficients()` (`sage.rings.function_field.differential.FunctionFieldDifferential` method), 272
`multiple()` (`sage.rings.function_field.jacobian_hess.JacobianPoint` method), 334
`multiple()` (`sage.rings.function_field.jacobian_khuri_makdisi.JacobianPoint` method), 347
`multiple()` (`sage.rings.function_field.khuri_makdisi.KhuriMakdisi_base` method), 348
`multiplicity()` (`sage.rings.function_field.divisor.FunctionFieldDivisor` method), 260

N

`negate()` (`sage.rings.function_field.khuri_makdisi.KhuriMakdisi_base` method), 348
`negate()` (`sage.rings.function_field.khuri_makdisi.KhuriMakdisi_small` method), 349
`ngens()` (`sage.rings.function_field.function_field_polymod.FunctionField_polymod` method), 89
`ngens()` (`sage.rings.function_field.function_field_rational.RationalFunctionField` method), 52
`ngens()` (`sage.rings.function_field.ideal.FunctionFieldIdeal_module` method), 202
`ngens()` (`sage.rings.function_field.order_polymod.FunctionFieldMaximalOrder_polymod` method), 181
`ngens()` (`sage.rings.function_field.order_polymod.FunctionFieldMaximalOrderInfinite_polymod` method), 171
`ngens()` (`sage.rings.function_field.order_rational.FunctionFieldMaximalOrder_rational` method), 149
`norm()` (`sage.rings.function_field.element.FunctionFieldElement` method), 119
`norm()` (`sage.rings.function_field.ideal_polymod.FunctionFieldIdeal_polymod` method), 232

`nth_root()` (`sage.rings.function_field.element_polymod.FunctionFieldElement_polymod` method), 137
`nth_root()` (`sage.rings.function_field.element_rational.FunctionFieldElement_rational` method), 131
`nth_root()` (`sage.rings.function_field.element.FunctionFieldElement` method), 120
`number_of_rational_places()` (`sage.rings.function_field.function_field_polymod.FunctionField_global` method), 64
`numerator()` (`sage.rings.function_field.divisor.FunctionFieldDivisor` method), 261
`numerator()` (`sage.rings.function_field.element_rational.FunctionFieldElement_rational` method), 132

O

`order()` (`sage.rings.function_field.function_field.FunctionField` method), 23
`order()` (`sage.rings.function_field.jacobian_base.JacobianGroup_finite_field_base` method), 322
`order()` (`sage.rings.function_field.jacobian_base.JacobianPoint_finite_field_base` method), 323
`order()` (`sage.rings.function_field.jacobian_hess.JacobianPoint` method), 335
`order_infinite()` (`sage.rings.function_field.function_field.FunctionField` method), 24
`order_infinite_with_basis()` (`sage.rings.function_field.function_field.FunctionField` method), 25
`order_with_basis()` (`sage.rings.function_field.function_field.FunctionField` method), 26

P

`p_radical()` (`sage.rings.function_field.order_polymod.FunctionFieldMaximalOrder_global` method), 173
`parent()` (`sage.rings.function_field.jacobian_base.JacobianGroup_base` method), 320
`place()` (`sage.rings.function_field.ideal.FunctionFieldIdeal` method), 194
`place()` (`sage.rings.function_field.valuation_ring.FunctionFieldValuationRing` method), 281
`place_below()` (`sage.rings.function_field.place_polymod.FunctionFieldPlace_polymod` method), 249
`place_infinite()` (`sage.rings.function_field.function_field_rational.RationalFunctionField_global` method), 56
`place_set()` (`sage.rings.function_field.function_field.FunctionField` method), 28
`places()` (`sage.rings.function_field.function_field_polymod.FunctionField_global` method), 65

`places()` (*sage.rings.function_field.function_field_rational.RationalFunctionField_global* method), 56
`places_above()` (*sage.rings.function_field.function_field_polymod.FunctionField_simple* method), 101
`places_finite()` (*sage.rings.function_field.function_field_polymod.FunctionField_global* method), 65
`places_finite()` (*sage.rings.function_field.function_field_rational.RationalFunctionField_global* method), 56
`places_infinite()` (*sage.rings.function_field.function_field_polymod.FunctionField_simple* method), 102
`PlaceSet` (class in *sage.rings.function_field.place*), 240
`point()` (*sage.rings.function_field.jacobian_hess.JacobianGroup* method), 330
`point()` (*sage.rings.function_field.jacobian_khuri_makdisi.JacobianGroup* method), 340
`poles()` (*sage.rings.function_field.element.FunctionFieldElement* method), 120
`polynomial()` (*sage.rings.function_field.function_field_polymod.FunctionField_polymod* method), 89
`polynomial()` (*sage.rings.function_field.order_basis.FunctionFieldOrder_basis* method), 162
`polynomial()` (*sage.rings.function_field.order_basis.FunctionFieldOrderInfinite_basis* method), 156
`polynomial()` (*sage.rings.function_field.order_polymod.FunctionFieldMaximalOrder_polymod* method), 182
`polynomial_ring()` (*sage.rings.function_field.function_field_polymod.FunctionField_polymod* method), 90
`polynomial_ring()` (*sage.rings.function_field.function_field_rational.RationalFunctionField* method), 52
`prime_below()` (*sage.rings.function_field.ideal_polymod.FunctionFieldIdeal_polymod* method), 233
`prime_below()` (*sage.rings.function_field.ideal_polymod.FunctionFieldIdealInfinite_polymod* method), 217
`prime_divisor()` (in module *sage.rings.function_field.divisor*), 263
`prime_ideal()` (*sage.rings.function_field.order_rational.FunctionFieldMaximalOrderInfinite_rational* method), 147
`prime_ideal()` (*sage.rings.function_field.place.FunctionFieldPlace* method), 240
`primitive_element()` (*sage.rings.function_field.function_field_polymod.FunctionField_polymod* method), 90

`primitive_integral_element_infinite()` (*sage.rings.function_field.function_field_polymod.FunctionField_integral* method), 68

R

`random_element()` (*sage.rings.function_field.function_field_polymod.FunctionField_polymod* method), 91
`random_element()` (*sage.rings.function_field.function_field_rational.RationalFunctionField* method), 52
`rank` (*sage.rings.function_field.jacobian_base.JacobianGroupFunctor* attribute), 318
`rational_function_field()` (*sage.rings.function_field.function_field.FunctionField* method), 29
`RationalFunctionField` (class in *sage.rings.function_field.function_field_rational*), 41
`RationalFunctionField_char_zero` (class in *sage.rings.function_field.function_field_rational*), 53
`RationalFunctionField_global` (class in *sage.rings.function_field.function_field_rational*), 54
`RationalFunctionFieldHigherDerivation_global` (class in *sage.rings.function_field.derivations_polymod*), 290
`relative_degree()` (*sage.rings.function_field.place_polymod.FunctionFieldPlace_polymod* method), 250
`residue()` (*sage.rings.function_field.differential.FunctionFieldDifferential* method), 273
`residue_field()` (*sage.rings.function_field.function_field_polymod.FunctionField_simple* method), 102
`residue_field()` (*sage.rings.function_field.function_field_rational.RationalFunctionField* method), 53
`residue_field()` (*sage.rings.function_field.place_polymod.FunctionFieldPlace_polymod* method), 250
`residue_field()` (*sage.rings.function_field.place_rational.FunctionFieldPlace_rational* method), 244
`residue_field()` (*sage.rings.function_field.valuation_ring.FunctionFieldValuationRing* method), 281
`reversed_hermite_form()` (in module *sage.rings.function_field.hermite_form_polynomial*), 352
`ring()` (*sage.rings.function_field.ideal.FunctionFieldIdeal* method), 197
`ring()` (*sage.rings.function_field.ideal.IdealMonoid* method), 203

S

[sage.rings.function_field.constructor module](#), 311
[sage.rings.function_field.derivations module](#), 283
[sage.rings.function_field.derivations_polymod module](#), 287
[sage.rings.function_field.derivations_rational module](#), 285
[sage.rings.function_field.differential module](#), 265
[sage.rings.function_field.divisor module](#), 253
[sage.rings.function_field.element module](#), 105
[sage.rings.function_field.element_polymod module](#), 135
[sage.rings.function_field.element_rational module](#), 127
[sage.rings.function_field.extensions module](#), 305
[sage.rings.function_field.function_field module](#), 3
[sage.rings.function_field.function_field_polymod module](#), 59
[sage.rings.function_field.function_field_rational module](#), 41
[sage.rings.function_field.hermite_form_polynomial module](#), 351
[sage.rings.function_field.ideal module](#), 183
[sage.rings.function_field.ideal_polymod module](#), 213
[sage.rings.function_field.ideal_rational module](#), 205
[sage.rings.function_field.jacobian_base module](#), 315
[sage.rings.function_field.jacobian_hess module](#), 327
[sage.rings.function_field.jacobian_khuri_makdisi module](#), 336
[sage.rings.function_field.khuri_makdisi module](#), 348
[sage.rings.function_field.maps module](#), 293
[sage.rings.function_field.order module](#), 139
[sage.rings.function_field.order_basis](#)

[module](#), 151
[sage.rings.function_field.order_polymod module](#), 165
[sage.rings.function_field.order_rational module](#), 145
[sage.rings.function_field.place module](#), 237
[sage.rings.function_field.place_polymod module](#), 247
[sage.rings.function_field.place_rational module](#), 243
[sage.rings.function_field.valuation_ring module](#), 279
[section\(\) \(sage.rings.function_field.maps.FractionFieldToFunctionField method\)](#), 294
[section\(\) \(sage.rings.function_field.maps.FunctionFieldToFractionField method\)](#), 299
[separable_model\(\) \(sage.rings.function_field.function_field_polymod.FunctionField_polymod method\)](#), 92
[set_base_place\(\) \(sage.rings.function_field.jacobian_base.Jacobian_base method\)](#), 326
[simple_model\(\) \(sage.rings.function_field.function_field_polymod.FunctionField_polymod method\)](#), 95
[some_elements\(\) \(sage.rings.function_field.function_field.FunctionField method\)](#), 30
[space_of_differentials\(\) \(sage.rings.function_field.function_field.FunctionField method\)](#), 31
[space_of_differentials_of_first_kind\(\) \(sage.rings.function_field.function_field.FunctionField method\)](#), 32
[space_of_holomorphic_differentials\(\) \(sage.rings.function_field.function_field.FunctionField method\)](#), 33
[sqrt\(\) \(sage.rings.function_field.element_rational.FunctionFieldElement_rational method\)](#), 132
[subs\(\) \(sage.rings.function_field.element.FunctionFieldElement method\)](#), 121
[subtract\(\) \(sage.rings.function_field.khuri_makdisi.KhuriMakdisi_base method\)](#), 348
[support\(\) \(sage.rings.function_field.divisor.FunctionFieldDivisor method\)](#), 261

T

[top\(\) \(sage.rings.function_field.extensions.ConstantFieldExtension method\)](#), 309
[trace\(\) \(sage.rings.function_field.element.FunctionFieldElement method\)](#), 123

V

[valuation\(\) \(sage.rings.function_field.differential.FunctionFieldDifferential method\)](#), 275

`valuation()` (*sage.rings.function_field.divisor.FunctionFieldDivisor* method), 262

`valuation()` (*sage.rings.function_field.element_rational.FunctionFieldElement_rational* method), 133

`valuation()` (*sage.rings.function_field.element.FunctionFieldElement* method), 124

`valuation()` (*sage.rings.function_field.function_field.FunctionField* method), 34

`valuation()` (*sage.rings.function_field.ideal_polymod.FunctionFieldIdeal_polymod* method), 235

`valuation()` (*sage.rings.function_field.ideal_polymod.FunctionFieldIdealInfinite_polymod* method), 218

`valuation()` (*sage.rings.function_field.ideal_rational.FunctionFieldIdeal_rational* method), 210

`valuation()` (*sage.rings.function_field.ideal_rational.FunctionFieldIdealInfinite_rational* method), 207

`valuation_ring()` (*sage.rings.function_field.place_polymod.FunctionFieldPlace_polymod* method), 251

`valuation_ring()` (*sage.rings.function_field.place_rational.FunctionFieldPlace_rational* method), 245

W

`weierstrass_places()` (*sage.rings.function_field.function_field_polymod.FunctionField_global* method), 66

Z

`zero()` (*sage.rings.function_field.jacobian_hess.JacobianGroup* method), 330

`zero()` (*sage.rings.function_field.jacobian_khuri_makdisi.JacobianGroup* method), 341

`zero_divisor()` (*sage.rings.function_field.khuri_makdisi.KhuriMakdisi_base* method), 349

`zeros()` (*sage.rings.function_field.element.FunctionFieldElement* method), 125