Discrete Valuations and Discrete Pseudo-Valuations

Release 10.4

The Sage Development Team

Jul 23, 2024
Valuations can be defined conveniently on some Sage rings such as p-adic rings and function fields.

### 1.1 p-adic valuations

Valuations on number fields can be easily specified if they uniquely extend the valuation of a rational prime:

```sage
sage: v = QQ.valuation(2)
sage: v(1024)
10
```

```python
>>> from sage.all import *

>>> v = QQ.valuation(Integer(2))

>>> v(Integer(1024))
10
```

They are normalized such that the rational prime has valuation 1:

```sage
sage: K.<a> = NumberField(x^2 + x + 1)
sage: v = K.valuation(2)
sage: v(1024)
10
```

```python
>>> from sage.all import *

>>> K = NumberField(x**Integer(2) + x + Integer(1), names=('a',)); (a,) = K._first_ngens(1)

>>> v = K.valuation(Integer(2))

>>> v(Integer(1024))
10
```

If there are multiple valuations over a prime, they can be obtained by extending a valuation from a smaller ring:

```sage
sage: K.<a> = NumberField(x^2 + x + 1)
sage: K.valuation(7)
Traceback (most recent call last):
...
ValueError: The valuation Gauss valuation induced by 7-adic valuation does not approximate a unique extension of 7-adic valuation with respect to x^2 + x + 1
```

```sage
sage: w,ww = QQ.valuation(7).extensions(K)
sage: w(a + 3), ww(a + 3)
(1, 0)
```

(continues on next page)
1.2 Valuations on Function Fields

Similarly, valuations can be defined on function fields:

```python
sage: K.<x> = FunctionField(QQ)
sage: v = K.valuation(x)
sage: v(1/x)
-1
sage: v = K.valuation(1/x)
sage: v(1/x)
1
```

On extensions of function fields, valuations can be created by providing a prime on the underlying rational function field when the extension is unique:

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

```python
>>> from sage.all import *

sage: K.<x> = FunctionField(QQ, names=('x',)); (x,) = K._first_ngens(1)
sage: R = K['y']; (y,) = R._first_ngens(1)
```
>>> L = K.extension(y**Integer(2) - x, names=('y',)); (y,) = L._first_ngens(1)
>>> v = L.valuation(x)
>>> v(x)
1

Valuations can also be extended from smaller function fields:

```
sage: K.<x> = FunctionField(QQ)
sage: v = K.valuation(x - 4)
sage: R.<y> = K[]
sage: L<y> = K.extension(y^2 - x)
sage: v.extensions(L)
[[ (x - 4)-adic valuation, v(y + 2) = 1 ]-adic valuation,
  [ (x - 4)-adic valuation, v(y - 2) = 1 ]-adic valuation]
```

```
>>> from sage.all import *
>>> K = FunctionField(QQ, names=('x',)); (x,) = K._first_ngens(1)
>>> v = K.valuation(x - Integer(4))
>>> R = K['y']; (y,) = R._first_ngens(1)
>>> L = K.extension(y**Integer(2) - x, names=('y',)); (y,) = L._first_ngens(1)
>>> v.extensions(L)
[[ (x - 4)-adic valuation, v(y + 2) = 1 ]-adic valuation,
  [ (x - 4)-adic valuation, v(y - 2) = 1 ]-adic valuation]
```
2.1 Mac Lane valuations

Internally, all the above is backed by the algorithms described in [Mac1936I] and [Mac1936II]. Let us consider the extensions of $K.\text{valuation}(x - 4)$ to the field $L$ above to outline how this works internally.

First, the valuation on $K$ is induced by a valuation on $Q[x]$. To construct this valuation, we start from the trivial valuation on $Q$ and consider its induced Gauss valuation on $Q[x]$, i.e., the valuation that assigns to a polynomial the minimum of the coefficient valuations:

```
sage: R.<x> = QQ[]
sage: v = GaussValuation(R, valuations.TrivialValuation(QQ))
```

The Gauss valuation can be augmented by specifying that $x - 4$ has valuation 1:

```
sage: v = v.augmentation(x - 4, 1); v
[ Gauss valuation induced by Trivial valuation on Rational Field, v(x - 4) = 1 ]
```

This valuation then extends uniquely to the fraction field:

```
sage: K.<x> = FunctionField(QQ)
sage: v = v.extension(K); v
(x - 4)-adic valuation
```

Over the function field we repeat the above process, i.e., we define the Gauss valuation induced by it and augment it to approximate an extension to $L$:
2.2 Limit valuations

In the previous example the final valuation \( \text{ww} \) is not merely given by evaluating \( w \) on the ring \( K[y] \):

\[
\text{sage: } \text{ww}(y^2 - x) \\
+\text{Infinity}
\]

\[
\text{sage: } y = R\text{.gen()}
\]

\[
\text{sage: } w(y^2 - x) \\
1
\]

Instead \( \text{ww} \) is given by a limit, i.e., an infinite sequence of augmentations of valuations:

\[
\text{sage: } \text{ww}_\text{.base_valuation} \\
[ \text{Gauss valuation induced by } (x - 4)\text{-adic valuation, } v(y - 2) = 1 , \ldots ]
\]

\[
\text{sage: } \text{ww}_\text{.base_valuation}.\text{approximation} \\
[ \text{Gauss valuation induced by } (x - 4)\text{-adic valuation, } v(y - 2) = 1 , \ldots ]
\]

The terms of this infinite sequence are computed on demand:

\[
\text{sage: } \text{ww}_\text{.base_valuation}.\text{approximation} \\
[ \text{Gauss valuation induced by } (x - 4)\text{-adic valuation, } v(y - 2) = 1 ]
\]

\[
\text{sage: } \text{ww}(y - 1/4*x - 1) \\
2
\]

\[
\text{sage: } \text{ww}_\text{.base_valuation}.\text{approximation} \\
[ \text{Gauss valuation induced by } (x - 4)\text{-adic valuation, } v(y + 1/64*x^2 - 3/8*x - 3/4) = -3 ]
\]
2.3 Non-classical valuations

Using the low-level interface we are not limited to classical valuations on function fields that correspond to points on the corresponding projective curves. Instead we can start with a non-trivial valuation on the field of constants:

```sage
sage: v = QQ.valuation(2)
sage: R.<x> = QQ[]
sage: w = GaussValuation(R, v) # v is not trivial
sage: K.<x> = FunctionField(QQ)
sage: w = w.extension(K)
sage: w.residue_field()
Rational function field in x over Finite Field of size 2
```

```sage
sage: v = QQ.valuation(Integer(2))
>>> R = QQ['x']; (x,) = R._first_ngens(1)
>>> w = GaussValuation(R, v) # v is not trivial
>>> K = FunctionField(QQ, names=('x',)); (x,) = K._first_ngens(1)
>>> w = w.extension(K)
>>> w.residue_field()
Rational function field in x over Finite Field of size 2
```
CHAPTER THREE

MAC LANE APPROXIMANTS

The main tool underlying this package is an algorithm by Mac Lane to compute, starting from a Gauss valuation on a polynomial ring and a monic squarefree polynomial $G$, approximations to the limit valuation which send $G$ to infinity:

```sage
def example():
    v = QQ.valuation(2)
    R.<x> = QQ[]
    f = x^5 + 3*x^4 + 5*x^3 + 8*x^2 + 6*x + 12
    v.mac_lane_approximants(f)  # random output (order may vary)
```

```python
>>> from sage.all import *
```

```python
>>> v = QQ.valuation(Integer(2))
```

```python
>>> R = QQ[x]; (x,) = R._first_ngens(1)
```

```python
>>> f = x**Integer(5) + Integer(3)*x**Integer(4) + Integer(5)*x**Integer(3) +
   Integer(8)*x**Integer(2) + Integer(6)*x + Integer(12)
```

```python
>>> v.mac_lane_approximants(f)  # random output (order may vary)
```

From these approximants one can already see the residual degrees and ramification indices of the corresponding extensions. The approximants can be pushed to arbitrary precision, corresponding to a factorization of $f$:

```sage
def example():
    v.mac_lane_approximants(f, required_precision=10)  # random output
```

```python
>>> from sage.all import *
```

```python
>>> v.mac_lane_approximants(f, required_precision=Integer(10))  # random output
```

9
The theory was originally described in [Mac1936I] and [Mac1936II]. A summary and some algorithmic details can also be found in Chapter 4 of [Rüt2014].
5.1 Value groups of discrete valuations

This file defines additive sub(semi-)groups of \( \mathbb{Q} \) and related structures.

AUTHORS:
- Julian Rüth (2013-09-06): initial version

EXAMPLES:

```python
sage: v = ZZ.valuation(2)
sage: v.value_group()
Additive Abelian Group generated by 1
sage: v.value_semigroup()
Additive Abelian Semigroup generated by 1
```

```python
>>> from sage.all import *
>>> from sage.rings.valuation.value_group import DiscreteValuationCodomain
>>> C = DiscreteValuationCodomain(); C
Codomain of Discrete Valuations
```

```python
class sage.rings.valuation.value_group.DiscreteValueGroup (generator)
    Bases: UniqueRepresentation, Parent

    The value group of a discrete valuation, an additive subgroup of \( \mathbb{Q} \) generated by generator.

    INPUT:
```
• generator—a rational number

Note: We do not rely on the functionality provided by additive abelian groups in Sage since these require the underlying set to be the integers. Therefore, we roll our own Z-module here. We could have used AdditiveAbelianGroupWrapper here, but it seems to be somewhat outdated. In particular, generic group functionality should now come from the category and not from the super-class. A facade of Q appeared to be the better approach.

EXAMPLES:

```sage```
```python
def example_usage():
    from sage.rings.valuation.value_group import DiscreteValueGroup
    sage: D1 = DiscreteValueGroup(0); D1
    Trivial Additive Abelian Group
    sage: D2 = DiscreteValueGroup(4/3); D2
    Additive Abelian Group generated by 4/3
    sage: D3 = DiscreteValueGroup(-1/3); D3
    Additive Abelian Group generated by 1/3
```

```sage```
```python
def example_usage():
    from sage.all import *
    from sage.rings.valuation.value_group import DiscreteValueGroup
    >>> D1 = DiscreteValueGroup(Integer(0)); D1
    Trivial Additive Abelian Group
    >>> D2 = DiscreteValueGroup(Integer(4)/Integer(3)); D2
    Additive Abelian Group generated by 4/3
    >>> D3 = DiscreteValueGroup(-Integer(1)/Integer(3)); D3
    Additive Abelian Group generated by 1/3
```

denominator()
Return the denominator of a generator of this group.

EXAMPLES:

```sage```
```python
def example_usage():
    from sage.rings.valuation.value_group import DiscreteValueGroup
    sage: DiscreteValueGroup(3/8).denominator()
    8
```

```sage```
```python
def example_usage():
    from sage.all import *
    from sage.rings.valuation.value_group import DiscreteValueGroup
    >>> DiscreteValueGroup(Integer(3)/Integer(8)).denominator()
    8
```

gen()
Return a generator of this group.

EXAMPLES:

```sage```
```python
def example_usage():
    from sage.rings.valuation.value_group import DiscreteValueGroup
    sage: DiscreteValueGroup(-3/8).gen()
    3/8
```

```sage```
```python
def example_usage():
    from sage.all import *
    from sage.rings.valuation.value_group import DiscreteValueGroup
    >>> DiscreteValueGroup(-Integer(3)/Integer(8)).gen()
    3/8
```
**index** (*other*)

Return the index of *other* in this group.

**INPUT:**

- *other* – a subgroup of this group

**EXAMPLES:**

```python
sage: from sage.rings.valuation.value_group import DiscreteValueGroup
sage: DiscreteValueGroup(3/8).index(DiscreteValueGroup(3))
8
sage: DiscreteValueGroup(3).index(DiscreteValueGroup(3/8))
Traceback (most recent call last):
... ValueError: other must be a subgroup of this group
sage: DiscreteValueGroup(3).index(DiscreteValueGroup(0))
Traceback (most recent call last):
... ValueError: other must have finite index in this group
sage: DiscreteValueGroup(0).index(DiscreteValueGroup(3))
Traceback (most recent call last):
... ValueError: other must be a subgroup of this group
```

```python
>>> from sage.all import *
>>> from sage.rings.valuation.value_group import DiscreteValueGroup
>>> DiscreteValueGroup(Integer(3)/Integer(8)).index(DiscreteValueGroup(Integer(3)))
8
>>> DiscreteValueGroup(Integer(3)).index(DiscreteValueGroup(Integer(3)/Integer(8)))
Traceback (most recent call last):
... ValueError: other must be a subgroup of this group
>>> DiscreteValueGroup(Integer(3)).index(DiscreteValueGroup(Integer(0)))
Traceback (most recent call last):
... ValueError: other must have finite index in this group
>>> DiscreteValueGroup(Integer(0)).index(DiscreteValueGroup(Integer(0)))
1
>>> DiscreteValueGroup(Integer(0)).index(DiscreteValueGroup(Integer(3)))
Traceback (most recent call last):
... ValueError: other must be a subgroup of this group
```

**is_trivial()**

Return whether this is the trivial additive abelian group.

**EXAMPLES:**

```python
sage: from sage.rings.valuation.value_group import DiscreteValueGroup
sage: DiscreteValueGroup(-3/8).is_trivial()
False
sage: DiscreteValueGroup(0).is_trivial()
True
```
Discrete Valuations and Discrete Pseudo-Valuations, Release 10.4

```python
>>> from sage.all import *
>>> from sage.rings.valuation.value_group import DiscreteValueGroup
>>> DiscreteValueGroup(-Integer(3)/Integer(8)).is_trivial()
False
>>> DiscreteValueGroup(Integer(0)).is_trivial()
True
```

def numerador():
    return the numerator of a generator of this group.

Examples:

```python
sage: from sage.rings.valuation.value_group import DiscreteValueGroup
sage: DiscreteValueGroup(3/8).numerator()
3
```

def some_elements():
    Return some typical elements in this group.

Examples:

```python
sage: from sage.rings.valuation.value_group import DiscreteValueGroup
sage: DiscreteValueGroup(-3/8).some_elements()
[3/8, -3/8, 0, 42, 3/2, -3/2, 9/8, -9/8]
```

class sage.rings.valuation.value_group.DiscreteValueSemigroup(generators):
    Bases: UniqueRepresentation, Parent

    The value semigroup of a discrete valuation, an additive subsemigroup of \( \mathbb{Q} \) generated by generators.

    INPUT:

    * generators – rational numbers

    Examples:

```python
sage: from sage.rings.valuation.value_group import DiscreteValueSemigroup
sage: D1 = DiscreteValueSemigroup(0); D1
Trivial Additive Abelian Semigroup
sage: D2 = DiscreteValueSemigroup(4/3); D2
Additive Abelian Semigroup generated by 4/3
sage: D3 = DiscreteValueSemigroup([-1/3, 1/2]); D3
Additive Abelian Semigroup generated by -1/3, 1/2
```

(continues on next page)
Discrete Valuations and Discrete Pseudo-Valuations, Release 10.4

Trivial Additive Abelian Semigroup

>>> D2 = DiscreteValueSemigroup(Integer(4)/Integer(3)); D2
Additive Abelian Semigroup generated by 4/3

>>> D3 = DiscreteValueSemigroup([-Integer(1)/Integer(3), Integer(1)/Integer(2)]);
D3
Additive Abelian Semigroup generated by -1/3, 1/2

gens()

Return the generators of this semigroup.

EXAMPLES:

sage: from sage.rings.valuation.value_group import DiscreteValueSemigroup
sage: DiscreteValueSemigroup(-3/8).gens()
(-3/8,)

is_group()

Return whether this semigroup is a group.

EXAMPLES:

sage: from sage.rings.valuation.value_group import DiscreteValueSemigroup
sage: DiscreteValueSemigroup(1).is_group()
False
sage: D = DiscreteValueSemigroup([-1, 1])
sage: D.is_group()
True

is_trivial()

Return whether this is the trivial additive abelian semigroup.

EXAMPLES:

5.1. Value groups of discrete valuations 17
Discrete Valuations and Discrete Pseudo-Valuations, Release 10.4

```python
sage: from sage.rings.valuation.value_group import DiscreteValueSemigroup
dsage: DiscreteValueSemigroup(-3/8).is_trivial()
False
dsage: DiscreteValueSemigroup([]).is_trivial()
True
```

```python
>>> from sage.all import *
>>> from sage.rings.valuation.value_group import DiscreteValueSemigroup

```

```python
>>> DiscreteValueSemigroup(-Integer(3)/Integer(8)).is_trivial()
False
>>> DiscreteValueSemigroup([]).is_trivial()
True
```

```python
some_elements()
Return some typical elements in this semigroup.

EXAMPLES:

```python
sage: from sage.rings.valuation.value_group import DiscreteValueSemigroup
dsage: list(DiscreteValueSemigroup([-3/8,1/2]).some_elements())

```

```python
>>> from sage.all import *
>>> from sage.rings.valuation.value_group import DiscreteValueSemigroup

```

```python
>>> list(DiscreteValueSemigroup([-Integer(3)/Integer(8),Integer(1)/Integer(2)]).some_elements())

```

5.2 Discrete valuations

This file defines abstract base classes for discrete (pseudo-)valuations.

AUTHORS:

• Julian Rüth (2013-03-16): initial version

EXAMPLES:

Discrete valuations can be created on a variety of rings:

```python
sage: ZZ.valuation(2)
2-adic valuation
sage: GaussianIntegers().valuation(3)  # needs sage.rings.number_field
3-adic valuation
sage: QQ.valuation(5)
5-adic valuation
sage: Zp(7).valuation()
7-adic valuation
```

```python
>>> from sage.all import *
>>> ZZ.valuation(Integer(2))
2-adic valuation
>>> GaussianIntegers().valuation(Integer(3))
```

(continues on next page)
3-adic valuation

```python
>>> QQ.valuation(Integer(5))
```

5-adic valuation

```python
>>> Zp(Integer(7)).valuation()
```

7-adic valuation

```python
sage:
K.<x> = FunctionField(QQ)
sage:
K.valuation(x)
(x)-adic valuation
sage:
K.valuation(x^2 + 1)
(x^2 + 1)-adic valuation
sage:
K.valuation(1/x)
```

Valuation at the infinite place

```python
>>> from sage.all import *
```

```python
sage: R.<x> = QQ[]
sage: v = QQ.valuation(2)
sage: w = GaussValuation(R, v)
sage: w.augmentation(x, 3)
```

[ Gauss valuation induced by 2-adic valuation, v(x) = 3 ]

```python
>>> from sage.all import *
```

```python
>>> R = QQ['x']; (x,) = R._first_ngens(1)
>>> v = QQ valuation(Integer(2))
>>> w = GaussValuation(R, v)
>>> w.augmentation(x, Integer(3))
```

[ Gauss valuation induced by 2-adic valuation, v(x) = 3 ]

We can also define discrete pseudo-valuations, i.e., discrete valuations that send more than just zero to infinity:

```python
sage: w.augmentation(x, infinity)
```

[ Gauss valuation induced by 2-adic valuation, v(x) = +Infinity ]

```python
>>> from sage.all import *
```

```python
>>> w.augmentation(x, infinity)
```

[ Gauss valuation induced by 2-adic valuation, v(x) = +Infinity ]

```python
class sage.rings.valuation.valuation.DiscretePseudovaluation (parent)
```

Bases: Morphism

Abstract base class for discrete pseudo-valuations, i.e., discrete valuations which might send more that just zero to infinity.

INPUT:
• **domain** – an integral domain

**EXAMPLES:**

```python
sage: v = ZZ.valuation(2); v  # indirect doctest
2-adic valuation

>>> from sage.all import *

>>> v = ZZ.valuation(Integer(2)); v  # indirect doctest
2-adic valuation
```

**is_equivalent**(f, g)

Return whether f and g are equivalent.

**EXAMPLES:**

```python
sage: v = QQ.valuation(2)
sage: v.is_equivalent(2, 1)
False
sage: v.is_equivalent(2, -2)
True
sage: v.is_equivalent(2, 0)
False
sage: v.is_equivalent(0, 0)
True

>>> from sage.all import *

>>> v = QQ.valuation(Integer(2))

>>> v.is_equivalent(Integer(2), Integer(1))
False

>>> v.is_equivalent(Integer(2), -Integer(2))
True

>>> v.is_equivalent(Integer(2), Integer(0))
False

>>> v.is_equivalent(Integer(0), Integer(0))
True
```

**class** `sage.rings.valuation.valuation.DiscreteValuation(parent)`

**Bases:** `DiscretePseudoValuation`

Abstract base class for discrete valuations.

**EXAMPLES:**

```python
sage: v = QQ.valuation(2)
sage: R.<x> = QQ[]
sage: v = GaussValuation(R, v)
sage: w = v.augmentation(x, 1337); w  # indirect doctest
[ Gauss valuation induced by 2-adic valuation, v(x) = 1337 ]

>>> from sage.all import *

>>> v = QQ.valuation(Integer(2))

>>> R = QQ['x']; (x,) = R._first_ngens(1)

>>> v = GaussValuation(R, v)

>>> w = v.augmentation(x, Integer(1337)); w  # indirect doctest
[ Gauss valuation induced by 2-adic valuation, v(x) = 1337 ]
```
**is_discrete_valuation()**

Return whether this valuation is a discrete valuation.

EXAMPLES:

```python
sage: v = valuations.TrivialValuation(ZZ)
sage: v.is_discrete_valuation()
True
```

```python
>>> from sage.all import *
>>> v = valuations.TrivialValuation(ZZ)
>>> v.is_discrete_valuation()
True
```

**mac_lane_approximant** \((G, valuation, approximants=None)\)

Return the approximant from `mac_lane_approximants()` for \(G\) which is approximated by \(valuation\).

**INPUT:**

- \(G\) – a monic squarefree integral polynomial in a univariate polynomial ring over the domain of this valuation
- \(valuation\) – a valuation on the parent of \(G\)
- \(approximants\) – the output of `mac_lane_approximants()` If not given, it is computed.

**EXAMPLES:**

```python
sage: v = QQ.valuation(2)
sage: R.<x> = QQ[]
sage: G = x^2 + 1
```

```python
>>> from sage.all import *
>>>
>>> v = QQ.valuation(Integer(2))
>>>
>>> R = QQ['x']; (x,) = R._first_ngens(1)
>>> G = x**Integer(2) + Integer(1)
```

We can select an approximant by approximating it:

```python
sage: w = GaussValuation(R, v).augmentation(x + 1, 1/2)
sage: v.mac_lane_approximant(G, w)
```

```python
# needs sage.geometry.polyhedron sage.rings.padics
[ Gauss valuation induced by 2-adic valuation, v(x + 1) = 1/2 ]
```

```python
>>> from sage.all import *
>>> w = GaussValuation(R, v).augmentation(x + Integer(1), Integer(1)/Integer(2))
>>> v.mac_lane_approximant(G, w)
```

```python
# needs sage.geometry.polyhedron sage.rings.padics
[ Gauss valuation induced by 2-adic valuation, v(x + 1) = 1/2 ]
```

As long as this is the only matching approximant, the approximation can be very coarse:

```python
sage: w = GaussValuation(R, v)
sage: v.mac_lane_approximant(G, w)
```

```python
# needs sage.geometry.polyhedron sage.rings.padics
[ Gauss valuation induced by 2-adic valuation, v(x + 1) = 1/2 ]
```
Discrete Valuations and Discrete Pseudo-Valuations, Release 10.4

>>> from sage.all import *
>>> w = GaussValuation(R, v)
>>> v.mac_lane_approximant(G, w)
# needs sage.geometry.polyhedron sage.rings.padics
[ Gauss valuation induced by 2-adic valuation, v(x + 1) = 1/2 ]

Or it can be very specific:

sage: w = GaussValuation(R, v).augmentation(x + 1, 1/2).augmentation(G, infinity)
# needs sage.geometry.polyhedron sage.rings.padics
[ Gauss valuation induced by 2-adic valuation, v(x + 1) = 1/2 ]

>>> from sage.all import *
>>> w = GaussValuation(R, v).augmentation(x + Integer(1), Integer(1)/Integer(2)).augmentation(G, infinity)
# needs sage.geometry.polyhedron sage.rings.padics
[ Gauss valuation induced by 2-adic valuation, v(x + 1) = 1/2 ]

But it must be an approximation of an approximant:

sage: w = GaussValuation(R, v).augmentation(x, 1/2)
sage: v.mac_lane_approximant(G, w)
Traceback (most recent call last):
... ValueError: The valuation
[ Gauss valuation induced by 2-adic valuation, v(x) = 1/2 ] is not an approximant for a valuation which extends 2-adic valuation with respect to x^2 + 1 since the valuation of x^2 + 1 does not increase in every step

>>> from sage.all import *
>>> w = GaussValuation(R, v).augmentation(x, Integer(1)/Integer(2))
>>> v.mac_lane_approximant(G, w)
Traceback (most recent call last):
... ValueError: The valuation
[ Gauss valuation induced by 2-adic valuation, v(x) = 1/2 ] is not an approximant for a valuation which extends 2-adic valuation with respect to x^2 + 1 since the valuation of x^2 + 1 does not increase in every step

The valuation must single out one approximant:

sage: G = x^2 - 1
sage: w = GaussValuation(R, v)
sage: v.mac_lane_approximant(G, w)
# needs sage.geometry.polyhedron sage.rings.padics
Traceback (most recent call last):
... ValueError: The valuation Gauss valuation induced by 2-adic valuation does not approximate a unique extension of 2-adic valuation with respect to x^2 - 1
sage: w = GaussValuation(R, v).augmentation(x + 1, 1)
(continues on next page)
sage: v.mac_lane_approximant(G, w)  # needs sage.geometry.polyhedron sage.rings.padics

Traceback (most recent call last):
...
ValueError: The valuation
[ Gauss valuation induced by 2-adic valuation, v(x + 1) = 1 ] does not
approximate a unique extension of 2-adic valuation with respect to x^2 - 1

sage: w = GaussValuation(R, v).augmentation(x + 1, 2)
sage: v.mac_lane_approximant(G, w)  # needs sage.geometry.polyhedron sage.rings.padics
[ Gauss valuation induced by 2-adic valuation, v(x + 1) = +Infinity ]

sage: w = GaussValuation(R, v).augmentation(x + 3, 2)
sage: v.mac_lane_approximant(G, w)  # needs sage.geometry.polyhedron sage.rings.padics
[ Gauss valuation induced by 2-adic valuation, v(x + 1) = 1 ]

>>> from sage.all import *
>>> G = x**Integer(2) - Integer(1)
>>> w = GaussValuation(R, v)
>>> v.mac_lane_approximant(G, w)  # needs sage.geometry.polyhedron sage.rings.padics

Traceback (most recent call last):
...
ValueError: The valuation Gauss valuation induced by 2-adic valuation
does not approximate a unique extension of 2-adic valuation
with respect to x^2 - 1

>>> w = GaussValuation(R, v).augmentation(x + Integer(1), Integer(1))
>>> v.mac_lane_approximant(G, w)  # needs sage.geometry.polyhedron sage.rings.padics
Traceback (most recent call last):
...
ValueError: The valuation
[ Gauss valuation induced by 2-adic valuation, v(x + 1) = 1 ] does not
approximate a unique extension of 2-adic valuation with respect to x^2 - 1

>>> w = GaussValuation(R, v).augmentation(x + Integer(1), Integer(2))
>>> v.mac_lane_approximant(G, w)  # needs sage.geometry.polyhedron sage.rings.padics
[ Gauss valuation induced by 2-adic valuation, v(x + 1) = +Infinity ]

>>> w = GaussValuation(R, v).augmentation(x + Integer(3), Integer(2))
>>> v.mac_lane_approximant(G, w)  # needs sage.geometry.polyhedron sage.rings.padics
[ Gauss valuation induced by 2-adic valuation, v(x + 1) = 1 ]

mac_lane_approximants (G, assume_squarefree=False, require_final_EF=True, required_precision=-1,
require_incomparability=False, require_maximal_degree=False,
algorithm='serial')

Return approximants on \( K[x] \) for the extensions of this valuation to \( L = K[x]/(G) \).

If \( G \) is an irreducible polynomial, then this corresponds to extensions of this valuation to the completion of \( L \).

INPUT:
• **G** – a monic squarefree integral polynomial in a univariate polynomial ring over the domain of this valuation

• **assume_squarefree** – a boolean (default: False), whether to assume that **G** is squarefree. If True, the squafreeness of **G** is not verified though it is necessary when **require_final_EF** is set for the algorithm to terminate.

• **require_final_EF** – a boolean (default: True); whether to require the returned key polynomials to be in one-to-one correspondence to the extensions of this valuation to **L** and require them to have the ramification index and residue degree of the valuations they correspond to.

• **required_precision** – a number or infinity (default: -1); whether to require the last key polynomial of the returned valuations to have at least that valuation.

• **require_incomparability** – a boolean (default: False); whether to require the returned valuations to be incomparable (with respect to the partial order on valuations defined by comparing them pointwise.)

• **require_maximal_degree** – a boolean (default: False); whether to require the last key polynomial of the returned valuation to have maximal degree. This is most relevant when using this algorithm to compute approximate factorizations of **G**, when set to True, the last key polynomial has the same degree as the corresponding factor.

• **algorithm** – one of "serial" or "parallel" (default: "serial"); whether or not to parallelize the algorithm

**EXAMPLES:**

```python
sage: v = QQ.valuation(2)
sage: R.<x> = QQ[]
sage: v.mac_lane_approximants(x^2 + 1)    # needs sage.geometry.polyhedron
[[ Gauss valuation induced by 2-adic valuation, v(x + 1) = 1/2 ]]

sage: v.mac_lane_approximants(x^2 + 1, required_precision=infinity)    # needs sage.geometry.polyhedron
[[ Gauss valuation induced by 2-adic valuation, v(x + 1) = 1/2,
  v(x^2 + 1) = +Infinity ]]

sage: v.mac_lane_approximants(x^2 + x + 1)
[[ Gauss valuation induced by 2-adic valuation, v(x^2 + x + 1) = +Infinity ]]
```

```python
>>> from sage.all import *
>>> v = QQ.valuation(Integer(2))
>>> R = QQ['x']; (x,) = R._first_ngens(1)
>>> v.mac_lane_approximants(x**Integer(2) + Integer(1))    # needs sage.geometry.polyhedron
[[ Gauss valuation induced by 2-adic valuation, v(x + 1) = 1/2 ]]

>>> v.mac_lane_approximants(x**Integer(2) + Integer(1), required_precision=infinity)    # needs sage.geometry.polyhedron
[[ Gauss valuation induced by 2-adic valuation, v(x + 1) = 1/2,
  v(x^2 + 1) = +Infinity ]]

>>> v.mac_lane_approximants(x**Integer(2) + x + Integer(1))
[[ Gauss valuation induced by 2-adic valuation, v(x^2 + x + 1) = +Infinity ]]
```

Note that **G** does not need to be irreducible. Here, we detect a factor **x + 1** and an approximate factor **x + 1** (which is an approximation to **x − 1**):

```python
sage: v.mac_lane_approximants(x^2 - 1)    # needs sage.geometry.polyhedron sage.rings.pads
```

(continues on next page)
Discrete valuations 25

However, it needs to be squarefree:

```sage
sage: v.mac_lane_approximants(x^2)
Traceback (most recent call last):
... 
ValueError: G must be squarefree
```

```sage
>>> from sage.all import *
>>>
>>> v.mac_lane_approximants(x**Integer(2))
Traceback (most recent call last):
... 
ValueError: G must be squarefree
```

`montes_factorization(G, assume_squarefree=False, required_precision=None)`

Factor $G$ over the completion of the domain of this valuation.

**INPUT:**

- $G$ — a monic polynomial over the domain of this valuation
- `assume_squarefree` — a boolean (default: False), whether to assume $G$ to be squarefree
- `required_precision` — a number or infinity (default: infinity); if infinity, the returned polynomials are actual factors of $G$, otherwise they are only factors with precision at least `required_precision`.

**ALGORITHM:**

We compute `mac_lane_approximants()` with `required_precision`. The key polynomials approximate factors of $G$. This can be very slow unless `required_precision` is set to zero. Single factor lifting could improve this significantly.

**EXAMPLES:**

```sage
sage: # needs sage.libs.ntl
sage: k = Qp(5,4)
sage: v = k.valuation()
sage: R.<x> = k[]
sage: G = x^2 + 1
sage: v.montes_factorization(G)  # needs sage.geometry.polyhedron
((1 + O(5^4))*x + 2 + 5 + 2*5^2 + 5^3 + O(5^4))
* ((1 + O(5^4))*x + 3 + 3*5 + 2*5^2 + 3*5^3 + O(5^4))
```

```sage
>>> from sage.all import *
>>> # needs sage.libs.ntl
>>> k = Qp(Integer(5),Integer(4))
>>> v = k.valuation()
```

(continues on next page)
Discrete Valuations and Discrete Pseudo-Valuations, Release 10.4

The computation might not terminate over incomplete fields (in particular because the factors can not be represented there):

sage: R.<x> = QQ[]
sage: v = QQ.valuation(2)
sage: v.montes_factorization(x^6 - 1) # not tested
(x - 1) * (x + 1) * (x^2 - x + 1) * (x^2 + x + 1)

sage: v.montes_factorization(x^7 - 1, required_precision=5) # needs sage.rings.padics
(x - 1) * (x^3 - 5*x^2 - 6*x - 1) * (x^3 + 6*x^2 + 5*x - 1)

REFERENCES:
The underlying algorithm is described in [Mac1936II] and thoroughly analyzed in [GMN2008].

class sage.rings.valuation.valuation.InfiniteDiscretePseudoValuation(parent)

Bases: DiscretePseudoValuation

Abstract base class for infinite discrete pseudo-valuations, i.e., discrete pseudo-valuations which are not discrete valuations.

EXAMPLES:

sage: v = QQ.valuation(2)
sage: R.<x> = QQ[]
sage: v = GaussValuation(R, v)
sage: w = v.augmentation(x, infinity); w # indirect doctest
[ Gauss valuation induced by 2-adic valuation, v(x) = +Infinity ]
is_discrete_valuation()

Return whether this valuation is a discrete valuation.

EXAMPLES:

```
sage: v = QQ.valuation(2)
sage: R.<x> = QQ[]
sage: v = GaussValuation(R, v)
sage: v.is_discrete_valuation()
True
sage: w = v.augmentation(x, infinity)
sage: w.is_discrete_valuation()
False
```
5.3 Spaces of valuations

This module provides spaces of exponential pseudo-valuations on integral domains. It currently only provides support for such valuations if they are discrete, i.e., their image is a discrete additive subgroup of the rational numbers extended by $\infty$.

AUTHORS:

- Julian Rüth (2016-10-14): initial version

EXAMPLES:
Note: Note that many tests not only in this module do not create instances of valuations directly since this gives the wrong inheritance structure on the resulting objects:

```
sage: from sage.rings.valuation.valuation_space import DiscretePseudoValuationSpace
sage: from sage.rings.valuation.trivial_valuation import TrivialDiscretePseudoValuation
sage: H = DiscretePseudoValuationSpace(QQ)
sage: v = TrivialDiscretePseudoValuation(H)
sage: v._test_category()
Traceback (most recent call last):
  ...  AssertionError: False is not true
```

Instead, the valuations need to be created through the `__make_element_class__` of the containing space:

```
sage: from sage.rings.valuation.trivial_valuation import TrivialDiscretePseudoValuation
sage: v = H.__make_element_class__(TrivialDiscretePseudoValuation)(H)
sage: v._test_category()
```

The factories such as `TrivialPseudoValuation` provide the right inheritance structure:

```
sage: v = valuations.TrivialPseudoValuation(QQ)
sage: v._test_category()
```

```
class sage.rings.valuation.valuation_space.DiscretePseudoValuationSpace(domain)
Bases: UniqueRepresentation, Homset

The space of discrete pseudo-valuations on domain.

EXAMPLES:
```
```
Note: We do not distinguish between the space of discrete valuations and the space of discrete pseudo-valuations. This is entirely for practical reasons: We would like to model the fact that every discrete valuation is also a discrete pseudo-valuation. At first, it seems to be sufficient to make sure that the \texttt{in} operator works which can essentially be achieved by overriding \texttt{_element_constructor_} of the space of discrete pseudo-valuations to accept discrete valuations by just returning them. Currently, however, if one does not change the parent of an element in \texttt{_element_constructor_} to \texttt{self}, then one cannot register that conversion as a coercion. Consequently, the operators \texttt{<=} and \texttt{>=} cannot be made to work between discrete valuations and discrete pseudo-valuations on the same domain (because the implementation only calls \texttt{_richcmp} if both operands have the same parent.) Of course, we could override \texttt{__ge__} and \texttt{__le__} but then we would likely run into other surprises. So in the end, we went for a single homspace for all discrete valuations (pseudo or not) as this makes the implementation much easier.

Todo: The comparison problem might be fixed by Issue \#22029 or similar.

\textbf{class ElementMethods}

\texttt{Bases: object}

Provides methods for discrete pseudo-valuations that are added automatically to valuations in this space.

\textbf{EXAMPLES:}

Here is an example of a method that is automagically added to a discrete valuation:

\begin{verbatim}
>>> from sage.all import *
>>> from sage.rings.valuation.valuation_space import DiscretePseudoValuationSpace
>>> H = DiscretePseudoValuationSpace(QQ)
>>> QQ.valuation(2)).is_discrete_pseudo_valuation() # indirect doctest
True
\end{verbatim}

The methods will be provided even if the concrete type is not created with \texttt{__make_element_class__}:

\begin{verbatim}
>>> from sage.rings.valuation.valuation_space import DiscretePseudoValuation
>>> m = DiscretePseudoValuation(H)
>>> m.parent() is H
True
\end{verbatim}
However, the category framework advises you to use inheritance:

```
sage: m._test_category()
AssertionError: False is not true
```

Using `__make_element_class__`, makes your concrete valuation inherit from this class:

```
sage: m = H.__make_element_class__(DiscretePseudoValuation)(H)
sage: m._test_category()
```

`change_domain(ring)`

Return this valuation over `ring`.

Unlike `extension()` or `restriction()`, this might not be completely sane mathematically. It is essentially a conversion of this valuation into another space of valuations.

**EXAMPLES:**

```
sage: v = QQ.valuation(3)
sage: v.change_domain(ZZ)
3-adic valuation
```

`element_with_valuation(s)`

Return an element in the domain of this valuation with valuation `s`.

**EXAMPLES:**

```
```
Discrete Valuations and Discrete Pseudo-Valuations, Release 10.4

sage: v = ZZ.valuation(2)
sage: v.element_with_valuation(10)
1024

>>> from sage.all import *
>>> v = ZZ.valuation(Integer(2))
>>> v.element_with_valuation(Integer(10))
1024

extension (ring)

Return the unique extension of this valuation to ring.

EXAMPLES:

sage: v = ZZ.valuation(2)
sage: w = v.extension(QQ)
sage: w.domain()
Rational Field

>>> from sage.all import *
>>> v = ZZ.valuation(Integer(2))
>>> w = v.extension(QQ)
>>> w.domain()
Rational Field

extensions (ring)

Return the extensions of this valuation to ring.

EXAMPLES:

sage: v = ZZ.valuation(2)
sage: v.extensions(QQ)
[2-adic valuation]

>>> from sage.all import *
>>> v = ZZ.valuation(Integer(2))
>>> v.extensions(QQ)
[2-adic valuation]

inverse (x, precision)

Return an approximate inverse of x.

The element returned is such that the product differs from 1 by an element of valuation at least precision.

INPUT:
  • x – an element in the domain of this valuation
  • precision – a rational or infinity

EXAMPLES:

sage: v = ZZ.valuation(2)
sage: x = 3
sage: y = v.inverse(3, 2); y
3
sage: x*y - 1
8
>>> from sage.all import *
>>> v = ZZ.valuation(Integer(2))
>>> x = Integer(3)
>>> y = v.inverse(Integer(3), Integer(2)); y
3
>>> x*y - Integer(1)
8

This might not be possible for elements of positive valuation:

```
sage: v.inverse(2, 2)
Traceback (most recent call last):
...  
ValueError: element has no approximate inverse in this ring
```

```
>>> from sage.all import *
>>> v.inverse(Integer(2), Integer(2))
Traceback (most recent call last):
...  
ValueError: element has no approximate inverse in this ring
```

Of course this always works over fields:

```
sage: v = QQ.valuation(2)
sage: v.inverse(2, 2)
1/2
```

```
>>> from sage.all import *
>>> v = QQ.valuation(Integer(2))
>>> v.inverse(Integer(2), Integer(2))
1/2
```

`is_discrete_pseudo_valuation()`

Return whether this valuation is a discrete pseudo-valuation.

**EXAMPLES:**

```
sage: QQ.valuation(2).is_discrete_pseudo_valuation()
True
```

```
>>> from sage.all import *
>>> QQ.valuation(Integer(2)).is_discrete_pseudo_valuation()
True
```

`is_discrete_valuation()`

Return whether this valuation is a discrete valuation, i.e., whether it is a discrete pseudo valuation that only sends zero to \( \infty \).

**EXAMPLES:**

```
sage: QQ.valuation(2).is_discrete_valuation()
True
```

```
>>> from sage.all import *
>>> QQ.valuation(Integer(2)).is_discrete_valuation()
True
```

5.3. Spaces of valuations
**is_negative_pseudo_valuation()**

Return whether this valuation is a discrete pseudo-valuation that does attain \(-\infty\), i.e., it is non-trivial and its domain contains an element with valuation \(\infty\) that has an inverse.

**EXAMPLES:**

```sage
sage: QQ.valuation(2).is_negative_pseudo_valuation()
False
```

```sage
>>> from sage.all import *
>>> QQ.valuation(Integer(2)).is_negative_pseudo_valuation()
False
```

**is_trivial()**

Return whether this valuation is trivial, i.e., whether it is constant \(\infty\) or constant zero for everything but the zero element.

Subclasses need to override this method if they do not implement `uniformizer()`.

**EXAMPLES:**

```sage
sage: QQ.valuation(7).is_trivial()
False
```

```sage
>>> from sage.all import *
>>> QQ.valuation(Integer(7)).is_trivial()
False
```

**lift(X)**

Return a lift of \(X\) in the domain which reduces down to \(X\) again via `reduce()`.

**EXAMPLES:**

```sage
sage: v = QQ.valuation(2)
sage: v.lift(v.residue_ring().one())
1
```

```sage
>>> from sage.all import *
>>> v = QQ.valuation(Integer(2))
>>> v.lift(v.residue_ring().one())
1
```

**lower_bound(x)**

Return a lower bound of this valuation at \(x\).

Use this method to get an approximation of the valuation of \(x\) when speed is more important than accuracy.

**EXAMPLES:**

```sage
sage: v = ZZ.valuation(2)
sage: v.lower_bound(2**10)
10
```

```sage
>>> from sage.all import *
>>> v = ZZ.valuation(Integer(2))
>>> v.lower_bound(Integer(2)**Integer(10))
10
```
reduce($x$)

Return the image of $x$ in the $\text{residue_ring()}$ of this valuation.

EXAMPLES:

```
sage: v = QQ.valuation(2)
sage: v.reduce(2) 0
sage: v.reduce(1) 1
sage: v.reduce(1/3) 1
sage: v.reduce(1/2) Traceback (most recent call last):
... ValueError: reduction is only defined for elements of non-negative valuation
```

residue_field()

Return the residue field of this valuation, i.e., the field of fractions of the $\text{residue_ring()}$, the elements of non-negative valuation modulo the elements of positive valuation.

EXAMPLES:

```
sage: QQ.valuation(2).residue_field() Finite Field of size 2
sage: valuations.TrivialValuation(QQ).residue_field() Rational Field
sage: valuations.TrivialValuation(ZZ).residue_field() Rational Field
sage: GaussValuation(ZZ['x'], ZZ.valuation(2)).residue_field() Rational function field in x over Finite Field of size 2
```

```
residue_ring()
Return the residue ring of this valuation, i.e., the elements of non-negative valuation modulo the elements of positive valuation. EXAMPLES:

```python
sage: QQ.valuation(2).residue_ring()
Finite Field of size 2
sage: valuations.TrivialValuation(QQ).residue_ring()
Rational Field
```

```
>>> from sage.all import *
>>> QQ.valuation(Integer(2)).residue_ring()
Finite Field of size 2
>>> valuations.TrivialValuation(QQ).residue_ring()
Rational Field
```

Note that a residue ring always exists, even when a residue field may not:

```python
sage: valuations.TrivialPseudoValuation(QQ).residue_ring()
Quotient of Rational Field by the ideal (1)
sage: valuations.TrivialValuation(ZZ).residue_ring()
Integer Ring
sage: GaussValuation(ZZ['x'], ZZ .valuation(2)).residue_ring()
Univariate Polynomial Ring in x over Finite Field of size 2...
```

>>> from sage.all import *
>>> valuations.TrivialPseudoValuation(QQ).residue_ring()
Quotient of Rational Field by the ideal (1)
>>> valuations.TrivialValuation(ZZ).residue_ring()
Integer Ring
>>> GaussValuation(ZZ['x'], ZZ .valuation(Integer(2))).residue_ring()
Univariate Polynomial Ring in x over Finite Field of size 2...
```

restriction(ring)
Return the restriction of this valuation to ring.

EXAMPLES:

```python
sage: v = QQ.valuation(2)
sage: w = v.restriction(ZZ)
sage: w.domain()
Integer Ring
```

```python
>>> from sage.all import *
>>> v = QQ.valuation(Integer(2))
>>> w = v.restriction(ZZ)
>>> w.domain()
Integer Ring
```

scale(scalar)
Return this valuation scaled by scalar.

INPUT:
- scalar – a non-negative rational number or infinity

EXAMPLES:
Discrete Valuations and Discrete Pseudo-Valuations, Release 10.4

```
sage: v = ZZ.valuation(3)
sage: w = v.scale(3)
sage: w(3)
3
>>> from sage.all import *
>>> v = ZZ.valuation(Integer(3))
>>> w = v.scale(Integer(3))
>>> w(Integer(3))
3
Scaling can also be done through multiplication with a scalar:

sage: w/3 == v
True
>>> from sage.all import *
>>> w/Integer(3) == v
True

Multiplication by zero produces the trivial discrete valuation:

sage: w = 0*v
sage: w(3)
0
sage: w(0)
+Infinity
>>> from sage.all import *
>>> w = Integer(0)*v
>>> w(Integer(3))
0
>>> w(Integer(0))
+Infinity

Multiplication by infinity produces the trivial discrete pseudo-valuation:

sage: w = infinity*v
sage: w(3)
+Infinity
sage: w(0)
+Infinity
>>> from sage.all import *
>>> w = infinity*v
>>> w(Integer(3))
+Infinity
>>> w(Integer(0))
+Infinity

separating_element(others)

Return an element in the domain of this valuation which has positive valuation with respect to this valuation but negative valuation with respect to the valuations in others.

EXAMPLES:

5.3. Spaces of valuations
shift \((x, s)\)

Shift \(x\) in its expansion with respect to \(\text{uniformizer()\ by s}\) “digits”.

For non-negative \(s\), this just returns \(x\) multiplied by a power of the uniformizer \(\pi\).

For negative \(s\), it does the same but when not over a field, it drops coefficients in the \(\pi\)-adic expansion which have negative valuation.

EXAMPLES:

\begin{verbatim}
sage: v = ZZ.valuation(2)
sage: v.shift(1, 10)
1024
sage: v.shift(11, -1)
5
\end{verbatim}

For some rings, there is no clear \(\pi\)-adic expansion. In this case, this method performs negative shifts by iterated division by the uniformizer and substraction of a lift of the reduction:

\begin{verbatim}
sage: R.<x> = ZZ[]
sage: v = ZZ.valuation(2)
sage: w = GaussValuation(R, v)
sage: w.shift(x, 1)
2*x
sage: w.shift(2*x, -1)
x
sage: w.shift(x + 2*x^2, -1)
x^2
\end{verbatim}

(continues on next page)
w.shift(x + Integer(2)*x**Integer(2), -Integer(1))

\[ x^2 \]

simplify \((x, \text{error=None, force=False})\)

Return a simplified version of \(x\).

Produce an element which differs from \(x\) by an element of valuation strictly greater than the valuation of \(x\) (or strictly greater than \(\text{error}\) if set.)

If \(\text{force}\) is not set, then expensive simplifications may be avoided.

**EXAMPLES:**

```python
sage: v = ZZ.valuation(2)
sage: v.simplify(6, force=True)
2
sage: v.simplify(6, error=0, force=True)
0
```

uniformizer()

Return an element in the domain which has positive valuation and generates the value group of this valuation.

**EXAMPLES:**

```python
sage: QQ.valuation(11).uniformizer()
11
```

Trivial valuations have no uniformizer:

```python
sage: from sage.rings.valuation.valuation_space import...
˓→DiscretePseudoValuationSpace
sage: v = DiscretePseudoValuationSpace(QQ).an_element()
sage: v.is_trivial()
True
sage: v.uniformizer()
Traceback (most recent call last):
...
ValueError: Trivial valuations do not define a uniformizing element
```
True
>>> v.uniformizer()
Traceback (most recent call last):
...
ValueError: Trivial valuations do not define a uniformizing element

upper_bound(x)
Return an upper bound of this valuation at x.
Use this method to get an approximation of the valuation of x when speed is more important than accuracy.

EXAMPLES:

```python
sage: v = ZZ.valuation(2)
sage: v.upper_bound(2^10)
10

>>> from sage.all import *
```n
```python
>>> v = ZZ.valuation(Integer(2))
```n
```python
>>> v.upper_bound(Integer(2)**Integer(10))
10
```n
value_group()
Return the value group of this discrete pseudo-valuation, the discrete additive subgroup of the rational numbers which is generated by the valuation of the uniformizer().

EXAMPLES:

```python
sage: QQ.valuation(2).value_group()
Additive Abelian Group generated by 1
```n
```python
>>> from sage.all import *
```n
```python
>>> QQ.valuation(Integer(2)).value_group()
Additive Abelian Group generated by 1
```n
A pseudo-valuation that is ∞ everywhere, does not have a value group:

```python
sage: from sage.rings.valuation.valuation_space import...
   DiscretePseudoValuationSpace
sage: v = DiscretePseudoValuationSpace(QQ).an_element()
sage: v.value_group()
Traceback (most recent call last):
...
ValueError: The trivial pseudo-valuation that is infinity everywhere does not have a value group.
```n
```python
>>> from sage.all import *
```n
```python
>>> from sage.rings.valuation.valuation_space import...
   DiscretePseudoValuationSpace
```n
```python
>>> v = DiscretePseudoValuationSpace(QQ).an_element()
```n
```python
>>> v.value_group()
Traceback (most recent call last):
...
ValueError: The trivial pseudo-valuation that is infinity everywhere does not have a value group.
```n
value_semigroup()

Return the value semigroup of this discrete pseudo-valuation, the additive subsemigroup of the rational numbers which is generated by the valuations of the elements in the domain.

EXAMPLES:

Most commonly, in particular over fields, the semigroup is the group generated by the valuation of the uniformizer:

```
sage: G = QQ.valuation(2).value_semigroup(); G
Additive Abelian Semigroup generated by -1, 1
sage: G in AdditiveMagmas().AdditiveAssociative().AdditiveUnital().˓→AdditiveInverse()
True
```

If the domain is a discrete valuation ring, then the semigroup consists of the positive elements of the `value_group()`:

```
sage: Zp(2).valuation().value_semigroup()
Additive Abelian Semigroup generated by 1
```

The semigroup can have a more complicated structure when the uniformizer is not in the domain:

```
sage: v = ZZ.valuation(2)
sage: R.<x> = ZZ[]
sage: w = GaussValuation(R, v)
sage: u = w.augmentation(x, 5/3)
sage: u.value_semigroup()
Additive Abelian Semigroup generated by 1, 5/3
```

```
>>> from sage.all import *
>>> G = QQ.valuation(Integer(2)).value_semigroup(); G
Additive Abelian Semigroup generated by -1, 1
>>> G in AdditiveMagmas().AdditiveAssociative().AdditiveUnital().˓→AdditiveInverse()
True
```

```
>>> from sage.all import *
>>> G = QQ.valuation(Integer(2)).value_semigroup(); G
Additive Abelian Semigroup generated by -1, 1
>>> G in AdditiveMagmas().AdditiveAssociative().AdditiveUnital().˓→AdditiveInverse()
True
```

```
>>> from sage.all import *
>>> from sage.all import *
>>> G = QQ.valuation(Integer(2)).value_semigroup(); G
Additive Abelian Semigroup generated by -1, 1
>>> G in AdditiveMagmas().AdditiveAssociative().AdditiveUnital().多重
True
```

```
>>> from sage.all import *
>>> G = QQ.valuation(Integer(2)).value_semigroup(); G
Additive Abelian Semigroup generated by -1, 1
>>> G in AdditiveMagmas().AdditiveAssociative().AdditiveUnital().多重
True
```

class sage.rings.valuation.valuation_space.ScaleAction

    Bases: Action

    Action of integers, rationals and the infinity ring on valuations by scaling it.

    EXAMPLES:
5.4 Trivial valuations

AUTHORS:

• Julian Rüth (2016-10-14): initial version

EXAMPLES:

```python
sage: v = valuations.TrivialValuation(QQ); v
Trivial valuation on Rational Field
sage: v(1)
0
```

```python
>>> from sage.all import *

>>> v = valuations.TrivialValuation(QQ); v
Trivial valuation on Rational Field

>>> v(Integer(1))
0
```

class sage.rings.valuation.trivial_valuation.TrivialDiscretePseudoValuation(parent)

Bases: TrivialDiscretePseudoValuation_base, InfiniteDiscretePseudoValuation

The trivial pseudo-valuation that is $\infty$ everywhere.

EXAMPLES:

```python
sage: v = valuations.TrivialPseudoValuation(QQ); v
Trivial pseudo-valuation on Rational Field

>>> from sage.all import *

>>> v = valuations.TrivialPseudoValuation(QQ); v
Trivial pseudo-valuation on Rational Field

lift(X)

Return a lift of $X$ to the domain of this valuation.

EXAMPLES:

```python
sage: v = valuations.TrivialPseudoValuation(QQ)
sage: v.lift(v.residue_ring().zero())
0
```
Discrete Valuations and Discrete Pseudo-Valuations, Release 10.4

```python
>>> from sage.all import *

>>> v = valuations.TrivialPseudoValuation(QQ)
>>> v.lift(v.residue_ring().zero())
0
```

**reduce**(x)

Reduce \(x\) modulo the positive elements of this valuation.

**EXAMPLES:**

```python
sage: v = valuations.TrivialPseudoValuation(QQ)
sage: v.reduce(1)
0
```

```python
>>> from sage.all import *

>>> v = valuations.TrivialPseudoValuation(QQ)
>>> v.reduce(Integer(1))
0
```

**residue_ring()**

Return the residue ring of this valuation.

**EXAMPLES:**

```python
sage: valuations.TrivialPseudoValuation(QQ).residue_ring()
Quotient of Rational Field by the ideal (1)
```

```python
>>> from sage.all import *

>>> valuations.TrivialPseudoValuation(QQ).residue_ring()
Quotient of Rational Field by the ideal (1)
```

**value_group()**

Return the value group of this valuation.

**EXAMPLES:**

A trivial discrete pseudo-valuation has no value group:

```python
sage: v = valuations.TrivialPseudoValuation(QQ)
sage: v.value_group()
Traceback (most recent call last):
...
ValueError: The trivial pseudo-valuation that is infinity everywhere does not have a value group.
```

```python
>>> from sage.all import *

>>> v = valuations.TrivialPseudoValuation(QQ)
>>> v.value_group()
Traceback (most recent call last):
...
ValueError: The trivial pseudo-valuation that is infinity everywhere does not have a value group.
```

class sage.rings.valuation.trivial_valuation.TrivialDiscretePseudoValuation_base(par-en)

Bases: DiscretePseudoValuation

5.4. Trivial valuations
Base class for code shared by trivial valuations.

EXAMPLES:

```sage
v = valuations.TrivialPseudoValuation(ZZ); v
Trivial pseudo-valuation on Integer Ring
```

```>>> from sage.all import *

>>> v = valuations.TrivialPseudoValuation(ZZ); v
Trivial pseudo-valuation on Integer Ring
```

`is_negative_pseudo_valuation()`

Return whether this valuation attains the value $-\infty$.

EXAMPLES:

```sage
v = valuations.TrivialPseudoValuation(QQ)
sage: v.is_negative_pseudo_valuation()
False
```

```>>> from sage.all import *

>>> v = valuations.TrivialPseudoValuation(QQ)

>>> v.is_negative_pseudo_valuation()
False
```

`is_trivial()`

Return whether this valuation is trivial.

EXAMPLES:

```sage
v = valuations.TrivialPseudoValuation(QQ)
sage: v.is_trivial()
True
```

```>>> from sage.all import *

>>> v = valuations.TrivialPseudoValuation(QQ)

>>> v.is_trivial()
True
```

`uniformizer()`

Return a uniformizing element for this valuation.

EXAMPLES:

```sage
v = valuations.TrivialPseudoValuation(ZZ)
sage: v.uniformizer()
Traceback (most recent call last):
  ...
ValueError: Trivial valuations do not define a uniformizing element
```

```>>> from sage.all import *

>>> v = valuations.TrivialPseudoValuation(ZZ)

>>> v.uniformizer()
Traceback (most recent call last):
  ...
ValueError: Trivial valuations do not define a uniformizing element
```
class sage.rings.valuation.trivial_valuation.TrivialDiscreteValuation

Bases: TrivialDiscretePseudoValuation_base, DiscreteValuation

The trivial valuation that is zero on non-zero elements.

EXAMPLES:

```python
sage: v = valuations.TrivialValuation(QQ); v
Trivial valuation on Rational Field

>>> from sage.all import *
>>>
>>> v = valuations.TrivialValuation(QQ); v
Trivial valuation on Rational Field
```

extensions (ring)

Return the unique extension of this valuation to ring.

EXAMPLES:

```python
sage: v = valuations.TrivialValuation(ZZ)
sage: v.extensions(QQ)
[Trivial valuation on Rational Field]

>>> from sage.all import *
>>>
>>> v = valuations.TrivialValuation(ZZ)
>>>
>>> v.extensions(QQ)
[Trivial valuation on Rational Field]
```

lift (X)

Return a lift of X to the domain of this valuation.

EXAMPLES:

```python
sage: v = valuations.TrivialValuation(QQ)
sage: v.lift(v.residue_ring().zero())
0

>>> from sage.all import *
>>>
>>> v = valuations.TrivialValuation(QQ)
>>>
>>> v.lift(v.residue_ring().zero())
0
```

reduce (x)

Reduce x modulo the positive elements of this valuation.

EXAMPLES:

```python
sage: v = valuations.TrivialValuation(QQ)
sage: v.reduce(1)
1

>>> from sage.all import *
>>>
>>> v = valuations.TrivialValuation(QQ)
>>>
>>> v.reduce(Integer(1))
1
```

5.4. Trivial valuations
residue_ring()  
Return the residue ring of this valuation.

EXAMPLES:

```
sage: valuations.TrivialValuation(QQ).residue_ring()
Rational Field
```

```
>>> from sage.all import *
>>> valuations.TrivialValuation(QQ).residue_ring()
Rational Field
```

value_group()  
Return the value group of this valuation.

EXAMPLES:

A trivial discrete valuation has a trivial value group:

```
sage: v = valuations.TrivialValuation(QQ)
sage: v.value_group()
Trivial Additive Abelian Group
```

```
>>> from sage.all import *
>>> v = valuations.TrivialValuation(QQ)
>>> v.value_group()
Trivial Additive Abelian Group
```

class sage.rings.valuation.trivial_valuation.TrivialValuationFactory(clazz, parent, *args, **kwargs)

Bases: UniqueFactory  
Create a trivial valuation on domain.

EXAMPLES:

```
sage: v = valuations.TrivialValuation(QQ); v
Trivial valuation on Rational Field
sage: v(1)
0
```

```
>>> from sage.all import *
>>> v = valuations.TrivialValuation(QQ); v
Trivial valuation on Rational Field
>>> v(Integer(1))
0
```

create_key(domain)  
Create a key that identifies this valuation.

EXAMPLES:

```
sage: valuations.TrivialValuation(QQ) is valuations.TrivialValuation(QQ) # indirect doctest
True
```
5.5 Gauss valuations on polynomial rings

This file implements Gauss valuations for polynomial rings, i.e. discrete valuations which assign to a polynomial the minimal valuation of its coefficients.

AUTHORS:

• Julian Rüth (2013-04-15): initial version

EXAMPLES:

A Gauss valuation maps a polynomial to the minimal valuation of any of its coefficients:

```
sage: R.<x> = QQ[]
sage: v0 = QQ.valuation(2)
sage: v = GaussValuation(R, v0); v
Gauss valuation induced by 2-adic valuation
sage: v(2*x + 2)
1
```

Gauss valuations can also be defined iteratively based on valuations over polynomial rings:

```
sage: v = v.augmentation(x, 1/4); v
[ Gauss valuation induced by 2-adic valuation, v(x) = 1/4 ]
sage: v = v.augmentation(x^4+2*x^3+2^2*x^2+2^2*x+2, 4/3); v
[ Gauss valuation induced by 2-adic valuation, v(x) = 1/4, v(x^4 + 2*x^3 + 2*x^2 +_ ˓→2*x + 2) = 4/3 ]
sage: S.<T> = R[
sage: w = GaussValuation(S, v); w
Gauss valuation induced by [ Gauss valuation induced by 2-adic valuation, v(x) = 1/4,␣
(continues on next page)
Discrete Valuations and Discrete Pseudo-Valuations, Release 10.4

\[ v(x^4 + 2x^3 + 2x^2 + 2x + 2) = 4/3 \]

\[ \text{sage: } w(2T + 1) \]

\[ 0 \]

```python
>>> from sage.all import *
>>> v = v.augmentation(x, Integer(1)/Integer(4)); v
[ Gauss valuation induced by 2-adic valuation, v(x) = 1/4 ]
>>> v = v.augmentation(x**Integer(4)+Integer(2)*x**Integer(3)+Integer(2)*x**Integer(2)+Integer(2)*x+Integer(2), Integer(4)/Integer(3)); v
[ Gauss valuation induced by 2-adic valuation, v(x) = 1/4, v(x^4 + 2x^3 + 2x^2 + 2x + 2) = 4/3 ]
>>> S = R['T']; (T,) = S._first_ngens(1)
>>> w = GaussValuation(S, v); w
Gauss valuation induced by Gauss valuation induced by 2-adic valuation, v(x) = 1/4, v(x^4 + 2x^3 + 2x^2 + 2x + 2) = 4/3]
>>> w(Integer(2)*T + Integer(1))
0
```

```python
class sage.rings.valuation.gauss_valuation.GaussValuationFactory

Bases: UniqueFactory

Create a Gauss valuation on \textit{domain}.

\textbf{INPUT:}

\begin{itemize}
  \item \texttt{domain} – a univariate polynomial ring
  \item \texttt{v} – a valuation on the base ring of \texttt{domain}, the underlying valuation on the constants of the polynomial ring (if unspecified take the natural valuation on the valued ring \texttt{domain}).
\end{itemize}

\textbf{EXAMPLES:}

The Gauss valuation is the minimum of the valuation of the coefficients:

\begin{verbatim}
sage: v = QQ.valuation(2)
sage: R.<x> = QQ[]
sage: w = GaussValuation(R, v)
sage: w(2)
1
sage: w(x)
0
sage: w(x + 2)
0
```

```python
>>> from sage.all import *
>>> R = QQ['x']; (x,) = R._first_ngens(1)
>>> w = GaussValuation(R, v)
>>> w(Integer(2))
1
>>> w(x)
0
>>> w(x + Integer(2))
0
```

\texttt{create_key} (\textit{domain}, \texttt{v=None})

Normalize and check the parameters to create a Gauss valuation.
create_object (version, key, **extra_args)

Create a Gauss valuation from normalized parameters.

class sage.rings.valuation.gauss_valuation.GaussValuation_generic (parent, v)
Bases: NonFinalInductiveValuation

A Gauss valuation on a polynomial ring domain.

INPUT:

• domain – a univariate polynomial ring over a valued ring \( R \)
• \( v \) – a discrete valuation on \( R \)

EXAMPLES:

```python
sage: R = Zp(3,5)
sage: S.<x> = R[]
# needs sage.libs.ntl
sage: v0 = R.valuation()
sage: v = GaussValuation(S, v0); v
Gauss valuation induced by 3-adic valuation
sage: S.<x> = QQ[]
sage: v = GaussValuation(S, QQ.valuation(5)); v
Gauss valuation induced by 5-adic valuation
```

\[E()\]

Return the ramification index of this valuation over its underlying Gauss valuation, i.e., 1.

EXAMPLES:

```python
>>> from sage.all import *
>>> R = Zp(Integer(3),Integer(5))
>>> S = R['x']; (x,) = S._first_ngens(1) # needs sage.libs.ntl
>>> v0 = R.valuation()
>>> v = GaussValuation(S, v0); v
# needs sage.libs.ntl
Gauss valuation induced by 3-adic valuation

>>> S = QQ['x']; (x,) = S._first_ngens(1)
>>> v = GaussValuation(S, QQ.valuation(Integer(5))); v
Gauss valuation induced by 5-adic valuation
```

5.5. Gauss valuations on polynomial rings 49
F()  
Return the degree of the residue field extension of this valuation over the Gauss valuation, i.e., 1.  

EXAMPLES:  

```python  
sage: # needs sage.libs.ntl  
sage: R.<u> = Qq(4,5)  
sage: S.<x> = R[]  
sage: v = GaussValuation(S)  
sage: v.F()  
1  
```

augmentation_chain()  
Return a list with the chain of augmentations down to the underlying Gauss valuation.  

EXAMPLES:  

```python  
sage: # needs sage.libs.ntl  
sage: R.<u> = Qq(4,5)  
sage: S.<x> = R[]  
sage: v = GaussValuation(S)  
sage: v.augmentation_chain()  
[Gauss valuation induced by 2-adic valuation]  
```

change_domain()  
Return this valuation as a valuation over `ring`.  

EXAMPLES:  

```python  
sage: v = ZZ.valuation(2)  
sage: R.<x> = ZZ[]  
sage: w = GaussValuation(R, v)  
sage: w.change_domain(QQ['x'])  
Gauss valuation induced by 2-adic valuation  
```

```python  
>>> from sage.all import *  
>>> # needs sage.libs.ntl  
>>> R = Qq(Integer(4),Integer(5), names=('u',)); (u,) = R._first_ngens(1)  
>>> S = R['x']; (x,) = S._first_ngens(1)  
>>> v = GaussValuation(S)  
>>> v.F()  
1  
```
**element_with_valuation**(*s*)

Return a polynomial of minimal degree with valuation *s*.

**EXAMPLES:**

```
sage: R.<x> = QQ[]
sage: v = GaussValuation(R, QQ.valuation(2))
sage: v.element_with_valuation(-2)
1/4
```

**equivalence_unit**(*s*, reciprocal=False)

Return an equivalence unit of valuation *s*.

**INPUT:**

- *s* – an element of the **value_group()**
- reciprocal – a boolean (default: False); whether or not to return the equivalence unit as the **equivalence_reciprocal()** of the equivalence unit of valuation -s

**EXAMPLES:**

```
sage: # needs sage.libs.ntl
sage: S.<x> = Qp(3,5)[]
sage: v = GaussValuation(S)
sage: v.equivalence_unit(2)
3^2 + O(3^7)
sage: v.equivalence_unit(-2)
3^-2 + O(3^3)
```

**extensions**(ring)

Return the extensions of this valuation to ring.

**EXAMPLES:**

```
sage: v = ZZ.valuation(2)
sage: R.<x> = ZZ[]
sage: w = GaussValuation(R, v)
sage: w.extensions(GaussianIntegers()]['x'])
[ Gauss valuation induced by 2-adic valuation]
```
Discrete Valuations and Discrete Pseudo-Valuations, Release 10.4

```python
>>> from sage.all import *
>>> v = ZZ.valuation(Integer(2))
>>> R = ZZ['x']; (x,) = R._first_ngens(1)
>>> w = GaussValuation(R, v)
>>> w.extensions(GaussianIntegers()['x'])
[# needs sage.rings.number_field
[Gauss valuation induced by 2-adic valuation]

\textbf{is\_gauss\_valuation()}\footnote{Return whether this valuation is a Gauss valuation.}

\textbf{is\_trivial()}\footnote{Return whether this is a trivial valuation (sending everything but zero to zero.)}

\textbf{lift}(F)\footnote{Return a lift of F.}

\textbf{INPUT:}

\begin{itemize}
  \item F – a polynomial over the \texttt{residue\_ring()} of this valuation
\end{itemize}

\textbf{OUTPUT:}

a (possibly non-monic) polynomial in the domain of this valuation which reduces to F

\textbf{EXAMPLES:}
```python
sage: # needs sage.libs.ntl
sage: S.<x> = Qp(3,5)[]
sage: v = GaussValuation(S)
sage: f = x^2 + 2*x + 16
sage: F = v.reduce(f); F
x^2 + 2*x + 1
sage: g = v.lift(F); g
(1 + O(3^5))*x^2 + (2 + O(3^5))*x + 1 + O(3^5)
sage: v.is_equivalent(f,g)
True
sage: g.parent() is v.domain()
True
```

See also:

`reduce()`

`lift_to_key(F)`

Lift the irreducible polynomial \( F \) from the `residue_ring()` to a key polynomial over this valuation.

**INPUT:**

- \( F \) – an irreducible non-constant monic polynomial in `residue_ring()` of this valuation

**OUTPUT:**

A polynomial \( f \) in the domain of this valuation which is a key polynomial for this valuation and which, for a suitable equivalence unit \( R \), satisfies that the reduction of \( Rf \) is \( F \)

**EXAMPLES:**

```python
sage: R.<u> = QQ
sage: S.<x> = R[[]]
sage: v = GaussValuation(S, QQ.valuation(Integer(2)))
sage: y = v.residue_ring().gen()
sage: f = v.lift_to_key(y**2 + y + Integer(1)); f
x^2 + x + 1
```

```python
>>> from sage.all import *
>>> # needs sage.libs.ntl
>>> S = Qp(Integer(3),Integer(5))['x']; (x,) = S._first_ngens(1)
>>> v = GaussValuation(S)
>>> f = x**Integer(2) + Integer(2)*x + Integer(16)
>>> F = v.reduce(f); F
x^2 + 2*x + 1
>>> g = v.lift(F); g
(1 + O(3^5))*x^2 + (2 + O(3^5))*x + 1 + O(3^5)
>>> v.is_equivalent(f,g)
True
>>> g.parent() is v.domain()
True
```

5.5. Gauss valuations on polynomial rings
**lower_bound**\((f)\)

Return a lower bound of this valuation at \(f\).

Use this method to get an approximation of the valuation of \(f\) when speed is more important than accuracy.

**EXAMPLES:**

```
sage: # needs sage.libs.ntl
sage: R.<u> = Qq(4, 5)
sage: S.<x> = R[]
sage: v = GaussValuation(S)
sage: v.lower_bound(1024*x + 2)
1
sage: v(1024*x + 2)
1
```

**monic_integral_model**\((G)\)

Return a monic integral irreducible polynomial which defines the same extension of the base ring of the domain as the irreducible polynomial \(G\) together with maps between the old and the new polynomial.

**EXAMPLES:**

```
sage: R.<x> = Qp(2, 5)[]
# needs sage.libs.ntl
sage: v = GaussValuation(R)
# needs sage.libs.ntl
sage: v.monic_integral_model(5*x^2 + 1/2*x + 1/4)
(Ring endomorphism of Univariate Polynomial Ring in x over 2-adic Field with capped relative precision 5
 Defn: (1 + O(2^5))*x |--> (2^-1 + O(2^4))*x,
Ring endomorphism of Univariate Polynomial Ring in x over 2-adic Field with capped relative precision 5
 Defn: (1 + O(2^5))*x |--> (2 + O(2^6))*x,
(1 + O(2^5))*x^2 + (1 + 2^2 + 2^3 + O(2^5))*x + 1 + 2^2 + 2^3 + O(2^5))
```

```python
>>> from sage.all import *
>>> # needs sage.libs.ntl
>>> R = Qp(Integer(2), Integer(5))
# needs sage.libs.ntl
>>> v = GaussValuation(R)
# needs sage.libs.ntl
>>> v.monic_integral_model(Integer(5)*x**2 + Integer(1)/Integer(2)*x + Integer(1)/Integer(4))
(Ring endomorphism of Univariate Polynomial Ring in x over 2-adic Field with capped relative precision 5
 Defn: (1 + O(2^5))*x |--> (2^-1 + O(2^4))*x,
Ring endomorphism of Univariate Polynomial Ring in x over 2-adic Field with capped relative precision 5
 Defn: (1 + O(2^5))*x |--> (2 + O(2^6))*x,
(1 + O(2^5))*x^2 + (1 + 2^2 + 2^3 + O(2^5))*x + 1 + 2^2 + 2^3 + O(2^5))
```
\textbf{Defn:} \((1 + O(2^5)) \times \mapsto (2 + O(2^6)) \times\),
\((1 + O(2^5)) \times x^2 + (1 + 2^2 + 2^3 + O(2^5)) \times + 1 + 2^2 + 2^3 + O(2^5))

\textbf{reduce} \((f, \text{check}=\text{True}, \text{degree\_bound}=\text{None}, \text{coefficients}=\text{None}, \text{valuations}=\text{None})\)

Return the reduction of \(f\) modulo this valuation.

\textbf{INPUT:}

- \(f\) – an integral element of the domain of this valuation
- \text{check} – whether or not to check whether \(f\) has non-negative valuation (default: \text{True})
- \text{degree\_bound} – an a-priori known bound on the degree of the result which can speed up the computation (default: not set)
- \text{coefficients} – the coefficients of \(f\) as produced by \text{coefficients()} or None (default: None); ignored
- \text{valuations} – the valuations of coefficients or None (default: None); ignored

\textbf{OUTPUT:}

A polynomial in the \text{residue\_ring()} of this valuation.

\textbf{EXAMPLES:}

\begin{verbatim}
 sage: # needs sage.libs.ntl
 sage: S.<x> = Qp(2,5)[]
 sage: v = GaussValuation(S)
 sage: f = x^2 + 2*x + 16
 sage: v.reduce(f)
 x^2
 sage: v.reduce(f).parent() is v.residue_ring()
 True

>>> from sage.all import *
 >>> # needs sage.libs.ntl
 >>> S = Qp(Integer(2),Integer(5))[x]; (x,) = S._first_ngens(1)
 >>> v = GaussValuation(S)
 >>> f = x**Integer(2) + Integer(2)*x + Integer(16)
 >>> v.reduce(f)
 x^2
 >>> v.reduce(f).parent() is v.residue_ring()
 True

The reduction is only defined for integral elements:

 sage: f = x^2/2
 # needs sage.libs.ntl
 sage: v.reduce(f)
 # needs sage.libs.ntl
 Traceback (most recent call last):
 ... ValueError: reduction not defined for non-integral elements and (2^\text{-}1 + O(2^\text{-}4)) \times x^2 is not integral over Gauss valuation induced by 2-adic valuation

>>> from sage.all import *
 >>> f = x**Integer(2)/Integer(2)

(continues on next page)
Discrete Valuations and Discrete Pseudo-Valuations, Release 10.4

```python
# needs sage.libs.ntl
v.reduce(f) # needs sage.libs.ntl
Traceback (most recent call last):
...
ValueError: reduction not defined for non-integral elements and (2^-1 + O(2^-4))*x^2 is not integral over Gauss valuation induced by 2-adic valuation
```

See also:

* lift ()

**residue_ring ()**

Return the residue ring of this valuation, i.e., the elements of valuation zero module the elements of positive valuation.

**EXAMPLES:**

```python
sage: S.<x> = Qp(2,5)[] # needs sage.libs.ntl
sage: v = GaussValuation(S) # needs sage.libs.ntl
sage: v.residue_ring() # needs sage.libs.ntl
Univariate Polynomial Ring in x over Finite Field of size 2 (using ...)
```

```python
>>> from sage.all import *
>>> S = Qp(Integer(2),Integer(5))['x']; (x,) = S._first_ngens(1) # needs sage.libs.ntl
>>> v = GaussValuation(S) # needs sage.libs.ntl
>>> v.residue_ring() # needs sage.libs.ntl
Univariate Polynomial Ring in x over Finite Field of size 2 (using ...)
```

**restriction (ring)**

Return the restriction of this valuation to ring.

**EXAMPLES:**

```python
sage: v = ZZ.valuation(2)
sage: R.<x> = ZZ[]
sage: w = GaussValuation(R, v)
sage: w.restriction(ZZ)
2-adic valuation
```

```python
>>> from sage.all import *
>>> v = ZZ.valuation(Integer(2))
>>> R = ZZ['x']; (x,) = R._first_ngens(1)
>>> w = GaussValuation(R, v)
>>> w.restriction(ZZ)
2-adic valuation
```

**scale (scalar)**

Return this valuation scaled by scalar.

**EXAMPLES:**

```python
```
sage: R.<x> = QQ[]
sage: v = GaussValuation(R, QQ.valuation(2))
sage: 3*v  # indirect doctest
Gauss valuation induced by 3 * 2-adic valuation

>>> from sage.all import *
>>> R = QQ['x']; (x,) = R._first_ngens(1)
>>> v = GaussValuation(R, QQ.valuation(Integer(2)))
>>> Integer(3)*v  # indirect doctest
Gauss valuation induced by 3 * 2-adic valuation

simplify (f, error=None, force=False, size_heuristic_bound=32, effective_degree=None, phiadic=True)

Return a simplified version of f.

Produce an element which differs from f by an element of valuation strictly greater than the valuation of f (or strictly greater than error if set.)

INPUT:

- f – an element in the domain of this valuation
- error – a rational, infinity, or None (default: None), the error allowed to introduce through the simplification
- force – whether or not to simplify f even if there is heuristically no change in the coefficient size of f expected (default: False)
- effective_degree – when set, assume that coefficients beyond effective_degree can be safely dropped (default: None)
- size_heuristic_bound – when force is not set, the expected factor by which the coefficients need to shrink to perform an actual simplification (default: 32)
- phiadic – whether to simplify in the x-adic expansion; the parameter is ignored as no other simplification is implemented

EXAMPLES:

sage: # needs sage.libs.ntl
sage: R.<u> = Qq(4, 5)
sage: S.<x> = R[]
sage: v = GaussValuation(S)
sage: f = x^10/2 + 1
sage: v.simplify(f)
(2^-1 + O(2^4))*x^10 + 1 + O(2^5)

>>> from sage.all import *
>>> # needs sage.libs.ntl
>>> R = Qq(Integer(4), Integer(5), names=('u',)); (u,) = R._first_ngens(1)
>>> S = R['x']; (x,) = S._first_ngens(1)
>>> v = GaussValuation(S)
>>> f = x**Integer(10)/Integer(2) + Integer(1)
>>> v.simplify(f)
(2^-1 + O(2^4))*x^10 + 1 + O(2^5)

uniformizer ()

Return a uniformizer of this valuation, i.e., a uniformizer of the valuation of the base ring.

EXAMPLES:
Discrete Valuations and Discrete Pseudo-Valuations, Release 10.4

```sage
sage: S.<x> = QQ[]
sage: v = GaussValuation(S, QQ.valuation(5))
sage: v.uniformizer()
5
sage: v.uniformizer().parent() is S
True
```

```python
>>> from sage.all import *
>>> S = QQ['x']; (x,) = S._first_ngens(1)
>>> v = GaussValuation(S, QQ.valuation(Integer(5)))
>>> v.uniformizer()
5
>>> v.uniformizer().parent() is S
True
```

**upper_bound** \((f)\)

Return an upper bound of this valuation at \(f\).

Use this method to get an approximation of the valuation of \(f\) when speed is more important than accuracy.

**EXAMPLES:**

```sage
sage: # needs sage.libsntl
sage: R.<u> = Qq(4, 5)
sage: S.<x> = R[]
sage: v = GaussValuation(S)
sage: v.upper_bound(1024*x + 1)
10
sage: v(1024*x + 1)
0
```

```python
>>> from sage.all import *
>>> # needs sage.libsntl
>>> R = Qq(Integer(4), Integer(5), names=('u',)); (u,) = R._first_ngens(1)
>>> S = R['x']; (x,) = S._first_ngens(1)
>>> v = GaussValuation(S)
>>> v.upper_bound(Integer(1024)*x + Integer(1))
10
>>> v(Integer(1024)*x + Integer(1))
0
```

**valuations** \((f, \text{coefficients=}\text{None, call_error=False})\)

Return the valuations of the \(f_i\phi^i\) in the expansion \(f = \sum f_i\phi^i\).

**INPUT:**

- \(f\) – a polynomial in the domain of this valuation
- \text{coefficients} – the coefficients of \(f\) as produced by \text{coefficients()} or None (default: None); this can be used to speed up the computation when the expansion of \(f\) is already known from a previous computation.
- \text{call_error} – whether or not to speed up the computation by assuming that the result is only used to compute the valuation of \(f\) (default: False)

**OUTPUT:**

A list, each entry a rational numbers or infinity, the valuations of \(f_0, f_1\phi, \ldots\)

**EXAMPLES:**
Discrete Valuations and Discrete Pseudo-Valuations, Release 10.4

```
sage: R = ZZ
sage: S.<x> = R[]
sage: v = GaussValuation(S, R.valuation(2))
sage: f = x^2 + 2*x + 16
sage: list(v.valuations(f))
[4, 1, 0]
```

```python
>>> from sage.all import *
>>> R = ZZ
>>> S = R[x]; (x,) = S._first_ngens(1)
>>> v = GaussValuation(S, R.valuation(Integer(2)))
>>> f = x**Integer(2) + Integer(2)*x + Integer(16)
>>> list(v.valuations(f))
[4, 1, 0]
```

**value_group()**

Return the value group of this valuation.

EXAMPLES:

```
sage: S.<x> = QQ[]
sage: v = GaussValuation(S, QQ.valuation(5))
sage: v.value_group()
Additive Abelian Group generated by 1
```

```python
>>> from sage.all import *
>>> S = QQ[x]; (x,) = S._first_ngens(1)
>>> v = GaussValuation(S, QQ.valuation(Integer(5)))
>>> v.value_group()
Additive Abelian Group generated by 1
```

**value_semigroup()**

Return the value semigroup of this valuation.

EXAMPLES:

```
sage: S.<x> = QQ[]
sage: v = GaussValuation(S, QQ.valuation(5))
sage: v.value_semigroup()
Additive Abelian Semigroup generated by -1, 1
```

```python
>>> from sage.all import *
>>> S = QQ[x]; (x,) = S._first_ngens(1)
>>> v = GaussValuation(S, QQ.valuation(Integer(5)))
>>> v.value_semigroup()
Additive Abelian Semigroup generated by -1, 1
```

5.5. Gauss valuations on polynomial rings 59
5.6 Valuations on polynomial rings based on $\phi$-adic expansions

This file implements a base class for discrete valuations on polynomial rings, defined by a $\phi$-adic expansion.

AUTHORS:


EXAMPLES:

The *Gauss valuation* is a simple example of a valuation that relies on $\phi$-adic expansions:

```python
sage: R.<x> = QQ[]
sage: v = GaussValuation(R, QQ.valuation(2))
```

```
>>> from sage.all import *
>>> R = QQ['x']; (x,) = R._first_ngens(1)
>>> v = GaussValuation(R, QQ.valuation(Integer(2)))
```

In this case, $\phi = x$, so the expansion simply lists the coefficients of the polynomial:

```python
sage: f = x^2 + 2*x + 2
sage: list(v.coefficients(f))
[2, 2, 1]
```

```
>>> from sage.all import *
>>> f = x**Integer(2) + Integer(2)*x + Integer(2)
>>> list(v.coefficients(f))
[2, 2, 1]
```

Often only the first few coefficients are necessary in computations, so for performance reasons, coefficients are computed lazily:

```python
sage: v.coefficients(f)
<generator object ...coefficients at 0x...>
```

```
>>> from sage.all import *
>>> v.coefficients(f)
<generator object ...coefficients at 0x...>
```

Another example of a *DevelopingValuation* is an *augmented valuation*:

```python
sage: w = v.augmentation(x^2 + x + 1, 3)
```

```
>>> from sage.all import *
>>> w = v.augmentation(x**Integer(2) + Integer(2)*x + Integer(2), Integer(3))
```

Here, the expansion lists the remainders of repeated division by $x^2 + x + 1$:

```python
sage: list(w.coefficients(f))
[x + 1, 1]
```

```
>>> from sage.all import *
>>> list(w.coefficients(f))
[x + 1, 1]
```
class sage.rings.valuation.developing_valuation.DevelopingValuation(parent, phi)

Bases: DiscretePseudoValuation

Abstract base class for a discrete valuation of polynomials defined over the polynomial ring domain by the \( \phi \)-adic development.

EXAMPLES:

```python
sage: R.<x> = QQ[]
sage: v = GaussValuation(R, QQ.valuation(7))
```

```python
>>> from sage.all import *

>>> R = QQ['x']; (x,) = R._first_ngens(1)

>>> v = GaussValuation(R, QQ.valuation(Integer(7)))
```

coefficients(f)

Return the \( \phi \)-adic expansion of \( f \).

INPUT:

- \( f \) – a monic polynomial in the domain of this valuation

OUTPUT:

An iterator \( f_0, f_1, \ldots, f_n \) of polynomials in the domain of this valuation such that \( f = \sum_i f_i \phi^i \)

EXAMPLES:

```python
sage: # needs sage.libs.ntl

sage: R = Qp(2,5)
sage: S.<x> = R[]
sage: v = GaussValuation(S)
sage: f = x^2 + 2*x + 3
sage: list(v.coefficients(f))  # note that these constants are in the...
--polynomial ring
[1 + 2 + O(2^5), 2 + O(2^6), 1 + O(2^5)]

sage: v = v.augmentation( x^2 + x + 1, 1)

sage: list(v.coefficients(f))
[(1 + O(2^5))*x + 2 + O(2^5), 1 + O(2^5)]
```

```python
>>> from sage.all import *

>>> # needs sage.libs.ntl

>>> R = Qp(Integer(2),Integer(5))

>>> S = R[x]; (x,) = S._first_ngens(1)

>>> v = GaussValuation(S)

>>> f = x**Integer(2) + Integer(2)*x + Integer(3)

>>> list(v.coefficients(f))  # note that these constants are in the...
--polynomial ring
[1 + 2 + O(2^5), 2 + O(2^6), 1 + O(2^5)]

>>> v = v.augmentation( x**Integer(2) + Integer(2)*x + Integer(3))

>>> list(v.coefficients(f))
[(1 + O(2^5))*x + 2 + O(2^5), 1 + O(2^5)]
```

effective_degree(f, valuations=None)

Return the effective degree of \( f \) with respect to this valuation.

The effective degree of \( f \) is the largest \( i \) such that the valuation of \( f \) and the valuation of \( f_i \phi^i \) in the development \( f = \sum_j f_j \phi^j \) coincide (see [Mac1936II] p.497.)

INPUT:
• \( f \) – a non-zero polynomial in the domain of this valuation

EXAMPLES:

```python
sage: # needs sage.libs.ntl
sage: R = Zp(2,5)
sage: S.<x> = R[]
sage: v = GaussValuation(S)
sage: v.effective_degree(x)
1
sage: v.effective_degree(2*x + 1)
0
```

```python
>>> from sage.all import *
>>> # needs sage.libs.ntl
>>> R = Zp(Integer(2),Integer(5))
>>> S = R['x']; (x,) = S._first_ngens(1)
>>> v = GaussValuation(S)
>>> v.effective_degree(x)
1
>>> v.effective_degree(Integer(2)*x + Integer(1))
0
```

newton_polygon \((f, valuations=None)\)

Return the Newton polygon of the \( \phi \)-adic development of \( f \).

INPUT:

• \( f \) – a polynomial in the domain of this valuation

EXAMPLES:

```python
sage: # needs sage.libs.ntl
sage: R = Qp(2,5)
sage: S.<x> = R[]
sage: v = GaussValuation(S)
sage: f = x^2 + 2*x + 3
sage: v.newton_polygon(f)
# needs sage.geometry.polyhedron
Finite Newton polygon with 2 vertices: (0, 0), (2, 0)
sage: v = v.augmentation( x^2 + x + 1, 1)

```

```python
>>> from sage.all import *
>>> # needs sage.libs.ntl
>>> R = Qp(Integer(2),Integer(5))
>>> S = R['x']; (x,) = S._first_ngens(1)
>>> v = GaussValuation(S)
>>> f = x**Integer(2) + Integer(2)*x + Integer(3)
```

(continues on next page)
### phi()

Return the polynomial $\phi$, the key polynomial of this valuation.

**EXAMPLES:**

```python
sage: R = Zp(2,5)
sage: S.<x> = R[]
    # needs sage.libs.ntl
sage: v = GaussValuation(S)
    # needs sage.libs.ntl
sage: v.phi()
(1 + O(2^5))*x
```

### valuations($f$)

Return the valuations of the $f_i\phi^i$ in the expansion $f = \sum f_i\phi^i$.

**INPUT:**

- $f$ – a polynomial in the domain of this valuation

**OUTPUT:**

A list, each entry a rational numbers or infinity, the valuations of $f_0, f_1\phi, \ldots$

**EXAMPLES:**

```python
sage: # needs sage.libs.ntl
sage: R = Qp(2,5)
sage: S.<x> = R[]
sage: v = GaussValuation(S, R.valuation())
sage: f = x**Integer(2) + Integer(2)*x + Integer(16)
sage: list(v.valuations(f))
[4, 1, 0]
```

(continues on next page)
5.7 Inductive valuations on polynomial rings

This module provides functionality for inductive valuations, i.e., finite chains of augmented valuations on top of a Gauss valuation.

AUTHORS:
- Julian Rüth (2016-11-01): initial version

EXAMPLES:

A Gauss valuation is an example of an inductive valuation:

```python
sage: R.<x> = QQ[]
sage: v = GaussValuation(R, QQ.valuation(2))
```

Generally, an inductive valuation is an augmentation of an inductive valuation, i.e., a valuation that was created from a Gauss valuation in a finite number of augmentation steps:

```python
sage: w = v.augmentation(x, 1)
sage: w = w.augmentation(x^2 + 2*x + 4, 3)
```

REFERENCES:
Inductive valuations are originally discussed in [Mac1936I] and [Mac1936II]. An introduction is also given in Chapter 4 of [Rütt2014].
extensions(other)

Return the extensions of this valuation to other.

EXAMPLES:

```python
sage: R.<x> = ZZ
sage: v = GaussValuation(R, ZZ.valuation(5))
sage: K.<x> = FunctionField(QQ)
sage: v.extensions(K)
[Trivial valuation on Rational Field]
```

class sage.rings.valuation.inductive_valuation.InductiveValuation(parent, phi)

Bases: DevelopingValuation

Abstract base class for iterated augmented valuations on top of a Gauss valuation.

EXAMPLES:

```python
sage: R.<x> = QQ
sage: v = GaussValuation(R, QQ.valuation(5))
```

E()

Return the ramification index of this valuation over its underlying Gauss valuation.

EXAMPLES:

```python
sage: # needs sage.libs.ntl
sage: R.<u> = Qq(4,5)
sage: S.<x> = R
sage: v = GaussValuation(S)
sage: v.E()
1
```

5.7. Inductive valuations on polynomial rings
**F()**

Return the residual degree of this valuation over its Gauss extension.

**EXAMPLES:**

```python
sage: # needs sage.libs.ntl
sage: R.<u> = Qq(4,5)
sage: S.<x> = R[
]
sage: v = GaussValuation(S)
sage: v.F()
1
```

**augmentation_chain()**

Return a list with the chain of augmentations down to the underlying Gauss valuation.

**EXAMPLES:**

```python
sage: # needs sage.libs.ntl
sage: R.<u> = Qq(4,5)
sage: S.<x> = R[
]
sage: v = GaussValuation(S)
sage: v.augmentation_chain()
[Gauss valuation induced by 2-adic valuation]
```

**element_with_valuation(s)**

Return a polynomial of minimal degree with valuation \( s \).

**EXAMPLES:**

```python
sage: R.<x> = QQ[
]
sage: v = GaussValuation(R, QQ.valuation(2))
sage: v.element_with_valuation(-2)
1/4
```

Depending on the base ring, an element of valuation \( s \) might not exist:
equivalence_reciprocal \( f, \text{coefficients=\text{None}, valuations=\text{None, check=\text{True}}} \)

Return an equivalence reciprocal of \( f \).

An equivalence reciprocal of \( f \) is a polynomial \( h \) such that \( f \cdot h \) is equivalent to 1 modulo this valuation (see [Mac1936II] p.497.)

INPUT:

- \( f \) – a polynomial in the domain of this valuation which is an equivalence_unit()
- \( \text{coefficients} \) – the coefficients of \( f \) in the phi()-adic expansion if known (default: None)
- \( \text{valuations} \) – the valuations of \( \text{coefficients} \) if known (default: None)
- \( \text{check} \) – whether or not to check the validity of \( f \) (default: True)

Warning: This method may not work over \( p \)-adic rings due to problems with the \text{gcd} implementation there.

EXAMPLES:

sage: \# needs sage.libs.ntl
sage: R = \text{Zp}(3,5)
sage: S.<x> = R[]
sage: v = GaussValuation(S)
sage: f = 3\cdot x + 2
sage: h = v.equivalence_reciprocal(f); h
2 + 0\cdot 3^5
sage: v.is_equivalent(f\cdot h, 1)
True

(continues on next page)
In an extended valuation over an extension field:

```python
sage: # needs sage.libs.ntl
sage: R.<u> = Qq(4,5)
sage: S.<x> = R[]
sage: v = GaussValuation(S)
sage: v = v.augmentation(x^2 + x + u, 1)
sage: f = 2*x + u
sage: h = v.equivalence_reciprocal(f); h
(u + 1) + O(2^5)
sage: v.is_equivalent(f*h, 1)
True
```

Extending the valuation once more:

```python
sage: # needs sage.libs.ntl
sage: v = v.augmentation((x^2 + x + u)^2 + 2*x*(x^2 + x + u) + 4*x, 3)
sage: h = v.equivalence_reciprocal(f); h
(u + 1) + O(2^5)
sage: v.is_equivalent(f*h, 1)
True
```

**equivalence_unit** \(s, \text{reciprocal}=\text{False}\)

Return an equivalence unit of valuation \(s\).

**INPUT:**

- \(s\) – an element of the \(\text{value_group()}

- \text{reciprocal} – a boolean (default: \text{False}); whether or not to return the equivalence unit as the \(\text{equivalence_reciprocal()}\) of the equivalence unit of valuation \(-s\).

**EXAMPLES:**
Discrete Valuations and Discrete Pseudo-Valuations, Release 10.4

sage: # needs sage.libs.ntl
sage: S.<x> = Qp(3,5)[]

sage: v = GaussValuation(S)
sage: v.equivalence_unit(2) 3^2 + O(3^7)
sage: v.equivalence_unit(-2) 3^-2 + O(3^3)

Note that this might fail for negative s if the domain is not defined over a field:

sage: v = ZZ.valuation(2)
sage: R.<x> = ZZ[]
sage: w = GaussValuation(R, v)
sage: w.equivalence_unit(1) 2
sage: w.equivalence_unit(-1)
Traceback (most recent call last):...
  ValueError: s must be in the value semigroup of this valuation
    but -1 is not in Additive Abelian Semigroup generated by 1

is_equivalence_unit (f, valuations=None)

Return whether the polynomial f is an equivalence unit, i.e., an element of effective_degree() zero (see [Mac1936II] p.497.)

INPUT:

• f – a polynomial in the domain of this valuation

EXAMPLES:

sage: # needs sage.libs.ntl
sage: R = Zp(2,5)
sage: S.<x> = R[]
sage: v = GaussValuation(S)
sage: v.is_equivalence_unit(x)
False
sage: v.is_equivalence_unit(S.zero())
False
sage: v.is_equivalence_unit(2*x + 1)
True

>>> from sage.all import *
>>> # needs sage.libs.ntl
>>> R = Zp(Integer(2),Integer(5))
>>> S = R['x']; (x,) = S._first_ngens(1)
>>> v = GaussValuation(S)
>>> v.is_equivalence_unit(x)
False
>>> v.is_equivalence_unit(S.zero())
False
>>> v.is_equivalence_unit(Integer(2)*x + Integer(1))
True

is_gauss_valuation()
Return whether this valuation is a Gauss valuation over the domain.

EXAMPLES:

sage: # needs sage.libs.ntl
sage: R.<u> = Qq(4,5)
sage: S.<x> = R[]
sage: v = GaussValuation(S)
sage: v.is_gauss_valuation()
True

>>> from sage.all import *
>>> R = Qq(Integer(4),Integer(5), names=('u',)); (u,) = R._first_ngens(1)
>>> S = R['x']; (x,) = S._first_ngens(1)
>>> v = GaussValuation(S)
>>> v.is_gauss_valuation()
True

monic_integral_model(G)
Return a monic integral irreducible polynomial which defines the same extension of the base ring of the domain as the irreducible polynomial G together with maps between the old and the new polynomial.

EXAMPLES:

sage: R.<x> = QQ[]
sage: v = GaussValuation(R, QQ.valuation(2))
sage: v.monic_integral_model(5*x^2 + 1/2*x + 1/4)
(Ring endomorphism of Univariate Polynomial Ring in x over Rational Field
Defn: x |---> 1/2*x,
Ring endomorphism of Univariate Polynomial Ring in x over Rational Field
Defn: x |---> 2*x,
x^2 + 1/5*x + 1/5)

>>> from sage.all import *
>>> R = QQ['x']; (x,) = R._first_ngens(1)
\begin{verbatim}
>>> v = GaussValuation(R, QQ.valuation(Integer(2)))
>>> v.monic_integral_model(Integer(5)*x**Integer(2) + Integer(1)/Integer(2)*x + Integer(1)/Integer(4))
(Ring endomorphism of Univariate Polynomial Ring in x over Rational Field
  Defn: x |--> 1/2*x,
  Ring endomorphism of Univariate Polynomial Ring in x over Rational Field
  Defn: x |--> 2*x,
  x^2 + 1/5*x + 1/5)

mu()

Return the valuation of \texttt{phi}().

EXAMPLES:

```
sage: R.<x> = QQ[]
sage: v = GaussValuation(R, QQ.valuation(2))
sage: v.mu()
0
```

```
>>> from sage.all import *

>>> R = QQ['x']; (x,) = R._first_ngens(1)
>>> v = GaussValuation(R, QQ.valuation(Integer(2)))
>>> v.mu()
0
```

class \texttt{sage.rings.valuation.inductive_valuation.InfiniteInductiveValuation}(parent, base_valuation)

Bases: \texttt{FinalInductiveValuation}, \texttt{InfiniteDiscretePseudovaluation}

Abstract base class for an inductive valuation which is not discrete, i.e., which assigns infinite valuation to its last key polynomial.

EXAMPLES:

```
sage: R.<x> = QQ[]
sage: v = GaussValuation(R, QQ.valuation(2))
sage: w = v.augmentation(x^2 + x + Integer(1), infinity)
```

```
>>> from sage.all import *

>>> R = QQ['x']; (x,) = R._first_ngens(1)
>>> v = GaussValuation(R, QQ.valuation(Integer(2)))
>>> w = v.augmentation(x^2 + x + Integer(1), infinity)
```

c\texttt{change\_domain}(ring)

Return this valuation over \texttt{ring}.

EXAMPLES:

We can turn an infinite valuation into a valuation on the quotient:

```
sage: R.<x> = QQ[]
sage: v = GaussValuation(R, QQ.valuation(2))
sage: w = v.augmentation(x^2 + x + Integer(1), infinity)
```
\end{verbatim}

5.7. Inductive valuations on polynomial rings
```python
sage: w.change_domain(R.quo(x^2 + x + 1))
2-adic valuation

>>> from sage.all import *
>>> R = QQ['x']; (x,) = R._first_ngens(1)
>>> v = GaussValuation(R, QQ.valuation(Integer(2)))
>>> w = v.augmentation(x**Integer(2) + x + Integer(1), infinity)
>>> w.change_domain(R.quo(x**Integer(2) + x + Integer(1)))
2-adic valuation
```

**class** `sage.rings.valuation.inductive_valuation.NonFinalInductiveValuation` *(parent, phi)*

**Bases:** `FiniteInductiveValuation, DiscreteValuation`

Abstract base class for iterated *augmented valuations* on top of a *Gauss valuation* which can be extended further through `augmentation()`.

**EXAMPLES:**

```python
sage: # needs sage.libs.ntl
sage: R.<u> = Qq(4,5)
sage: S.<x> = R[]
sage: v = GaussValuation(S)
sage: v = v.augmentation(x^2 + x + u, 1)
```

```python
>>> from sage.all import *
>>> # needs sage.libs.ntl
>>> R = Qq(Integer(4),Integer(5), names=('u',)); (u,) = R._first_ngens(1)
>>> S = R['x']; (x,) = S._first_ngens(1)
>>> v = GaussValuation(S)
>>> v = v.augmentation(x**Integer(2) + x + u, Integer(1))
```

**augmentation**(phi, mu, check=True)

Return the inductive valuation which extends this valuation by mapping phi to mu.

**INPUT:**

- phi – a polynomial in the domain of this valuation; this must be a key polynomial, see `is_key()` for properties of key polynomials.
- mu – a rational number or infinity, the valuation of phi in the extended valuation
- check – a boolean (default: True), whether or not to check the correctness of the parameters

**EXAMPLES:**

```python
sage: # needs sage.libs.ntl
sage: R.<u> = Qq(4,5)
sage: S.<x> = R[]
sage: v = GaussValuation(S)
sage: v = v.augmentation(x^2 + x + u, 1)
```

```python
>>> from sage.all import *
>>> # needs sage.libs.ntl
>>> R = Qq(Integer(4),Integer(5), names=('u',)); (u,) = R._first_ngens(1)
>>> S = R['x']; (x,) = S._first_ngens(1)
>>> v = GaussValuation(S)
>>> v = v.augmentation(x**Integer(2) + x + u, Integer(1))
```

[ Gauss valuation induced by 2-adic valuation,
\[ v((1 + O(2^5))*x^2 + (1 + O(2^5))*x + u + O(2^5)) = 1, \]
\[ v((1 + O(2^5))*x^4 \]
\[ + (2^2 + O(2^6))x^3 + (1 + (u + 1)*2 + O(2^5))x^2 + ((u + 1)*2^2 + O(2^6))x + (u + 1) + (u + 1)*2 + (u + 1)*2^2 + (u + 1)*2^3 + (u + 1)*2^4 + O(2^5) \]
\[ \Rightarrow 3 \]

```python
>>> from sage.all import *
>>> # needs sage.libs.ntl
>>> R = Qq(Integer(4),Integer(5), names=('u',)); (u,) = R._first_ngens(1)
>>> S = R['x']; (x,) = S._first_ngens(1)
>>> v = GaussValuation(S)
>>> v = v.augmentation(x**Integer(2) + x + u, Integer(1))
>>> v = v.augmentation((x**Integer(2) + x + u)**Integer(2) +
                  Integer(2)*x*(x**Integer(2) + x + u) + Integer(4)*x, Integer(3))
>>> v
[ Gauss valuation induced by 2-adic valuation,
  v((1 + O(2^5))*x^2 + (1 + O(2^5))*x + u + O(2^5)) = 1,
  v((1 + O(2^5))*x^4 + (2^2 + O(2^6))*x^3 + (1 + (u + 1)*2 + O(2^5))*x^2 + ((u + 1)*2^2 + O(2^6))x + (u + 1) + (u + 1)*2 + (u + 1)*2^2 + (u + 1)*2^3 + (u + 1)*2^4 + O(2^5)) \]
\[ \Rightarrow 3 \]
```

See also:

 augmented_valuation

equivalence_decomposition \( f, assume_not_equivalence_unit=False, coefficients=None, 
valuations=None, compute_unit=True, degree_bound=None \)

Return an equivalence decomposition of \( f \), i.e., a polynomial \( g(x) = e(x)\prod\phi_i(x) \) with \( e(x) \) an equivalence unit \( R \) and the \( \phi_i \) key polynomials such that \( f \) is_equivalent() to \( g \).

INPUT:

- \( f \) – a non-zero polynomial in the domain of this valuation
- \( assume_not_equivalence_unit \) – whether or not to assume that \( f \) is not an equivalence unit (default: False)
- \( coefficients \) – the coefficients of \( f \) in the \( \phi \)-adic expansion if known (default: None)
- \( valuations \) – the valuations of \( coefficients \) if known (default: None)
- \( compute_unit \) – whether or not to compute the unit part of the decomposition (default: True)
- \( degree_bound \) – a bound on the degree of the \( _\text{equivalence\_reduction}() \) of \( f \) (default: None)

ALGORITHM:

We use the algorithm described in Theorem 4.4 of [Mac1936II]. After removing all factors \( \phi \) from a polynomial \( f \), there is an equivalence unit \( R \) such that \( Rf \) has valuation zero. Now \( Rf \) can be factored as \( \prod_{i} \alpha_i \) over the \( \text{residue\_field}() \). Lifting all \( \alpha_i \) to key polynomials \( \phi_i \) gives \( Rf = \prod_{i} \alpha_i \phi_i \) for suitable equivalence units \( R_i \) (see \( \text{lift\_to\_key}() \)). Taking \( R' \) an equivalence_reciprocal() of \( R \), we have \( f \) equivalent to \( (R' \prod_{i} \alpha_i) \prod_{i} \phi_i \).

EXAMPLES:
A polynomial that is an equivalence unit, is returned as the unit part of a Factorization, leading to a unit non-minimal degree:

```
sage: w = v.augmentation(x, 1)  # needs sage.libs.ntl

sage: F = w.equivalence_decomposition(x^2+1); F  # needs sage.libs.ntl
(1 + O(2^10))*x^2 + 1 + O(2^10)

sage: F.unit()  # needs sage.libs.ntl
(1 + O(2^10))*x^2 + 1 + O(2^10)
```

However, if the polynomial has a non-unit factor, then the unit might be replaced by a factor of lower degree:

```
sage: f = x * (x^2 + 1)  # needs sage.libs.ntl
```

(continues on next page)
Examples over an iterated unramified extension:

```python
sage: # needs sage.libsntl
sage: v = v.augmentation(x^2 + x + u, 1)
sage: v = v.augmentation((x^2 + x + u)^2 + 2*x*(x^2 + x + u) + 4*x, 3)
sage: v.equivalence_decomposition(x)
(1 + O(2^10))*x
sage: F = v.equivalence_decomposition( v.phi() )
sage: len(F)
1
sage: F = v.equivalence_decomposition( v.phi() * (x^4 + 4*x^3 + (7 + 2*u)*x^2 + (8 + 4*u)*x + 1023 + 3*u) )
```

```python
>>> from sage.all import *
```
INPUT:

- $f$ – a non-constant polynomial in the domain of this valuation

EXAMPLES:

```python
sage: # needs sage.libsntl
sage: R.<u> = Qq(4,5)
sage: S.<x> = R[]
sage: v = GaussValuation(S)
sage: v.is_equivalence_irreducible(x)
True
sage: v.is_equivalence_irreducible(x^2)
False
sage: v.is_equivalence_irreducible(x^2 + 2)
False

>>> from sage.all import *
>>> # needs sage.libsntl

R = Qq(Integer(4), Integer(5), names=('u',)); (u,) = R._first_ngens(1)
S = R['x']; (x,) = S._first_ngens(1)
v = GaussValuation(S)
>>> v.is_equivalence_irreducible(x)
True
>>> v.is_equivalence_irreducible(x**Integer(2))
False
>>> v.is_equivalence_irreducible(x**Integer(2) + Integer(2))
False
```

```

is_key(phi, explain=False, assume_equivalence_irreducible=False)
```

Return whether $\phi$ is a key polynomial for this valuation, i.e., whether it is monic, whether it is `is_equivalence_irreducible()`, and whether it is `is_minimal()`.

INPUT:

- $\phi$ – a polynomial in the domain of this valuation

- explain – a boolean (default: False), if True, return a string explaining why $\phi$ is not a key polynomial

EXAMPLES:

```
sage: # needs sage.libsntl
sage: R.<u> = Qq(4, 5)
sage: S.<x> = R[]
sage: v = GaussValuation(S)
sage: v.is_key(x)
True
sage: v.is_key(2*x, explain=True)
(False, 'phi must be monic')
sage: v.is_key(x^2, explain=True)
(False, 'phi must be equivalence irreducible')
sage: w = v.augmentation(x, 1)
sage: w.is_key(x + 1, explain = True)
(False, 'phi must be minimal')
```

```

>>> from sage.all import *
>>> # needs sage.libsntl

R = Qq(Integer(4), Integer(5), names=('u',)); (u,) = R._first_ngens(1)
(continues on next page)
```
is_minimal \( (f, \text{assume\_equivalence\_irreducible=}\text{False}) \)

Return whether the polynomial \( f \) is minimal with respect to this valuation.

A polynomial \( f \) is minimal with respect to \( v \) if it is not a constant and any non-zero polynomial \( h \) which is \( v \)-divisible by \( f \) has at least the degree of \( f \).

A polynomial \( h \) is \( v \)-divisible by \( f \) if there is a polynomial \( c \) such that \( fc \text{ is\_equivalent()} \) to \( h \).

ALGORITHM:
Based on Theorem 9.4 of [Mac1936II].

EXAMPLES:

```python
sage: # needs sage.libs.ntl
sage: R.<u> = Qq(4, 5)
sage: S.<x> = R[]
sage: v = GaussValuation(S)
sage: v.is_minimal(x + 1)
True
sage: w = v.augmentation(x, 1)
sage: w.is_minimal(x + 1)
False
```

lift_to_key \( (F) \)

Lift the irreducible polynomial \( F \) from the residue\_ring() to a key polynomial over this valuation.

INPUT:

- \( F \) – an irreducible non-constant monic polynomial in residue\_ring() of this valuation

OUTPUT:

A polynomial \( f \) in the domain of this valuation which is a key polynomial for this valuation and which is such that an augmentation() with this polynomial adjoins a root of \( F \) to the resulting residue\_ring().

5.7. Inductive valuations on polynomial rings 77
More specifically, if $F$ is not the generator of the residue ring, then multiplying $f$ with the `equivalence_reciprocal()` of the `equivalence_unit()` of the valuation of $f$, produces a unit which reduces to $F$.

**EXAMPLES:**

```python
sage: # needs sage.libs.ntl
sage: R.<u> = Qq(4,10)
sage: S.<x> = R[]
sage: v = GaussValuation(S)
sage: y = v.residue_ring().gen()
sage: u0 = v.residue_ring().base_ring().gen()
sage: f = v.lift_to_key(y^2 + y + u0); f
(1 + O(2^10))*x^2 + (1 + O(2^10))*x + u + O(2^10)
```

```
>>> from sage.all import *
>>> # needs sage.libs.ntl
>>> R = Qq(Integer(4),Integer(10), names=('u',)); (u,) = R._first_ngens(1)
>>> S = R['x']; (x,) = S._first_ngens(1)
>>> v = GaussValuation(S)
>>> y = v.residue_ring().gen()
>>> u0 = v.residue_ring().base_ring().gen()
>>> f = v.lift_to_key(y**Integer(2) + y + u0); f
(1 + O(2^10))*x^2 + (1 + O(2^10))*x + u + O(2^10)
```

**mac_lane_step** ($G$, principal_part_bound=None, assume_squarefree=False, assume_equivalence_irreducible=False, report_degree_bounds_and_caches=False, coefficients=None, valuations=None, check=True, allow_equivalent_key=True)

Perform an approximation step towards the squarefree monic non-constant integral polynomial $G$ which is not an equivalence unit.

This performs the individual steps that are used in `mac_lane_approximants()`.

**INPUT:**

- $G$ – a squarefree monic non-constant integral polynomial $G$ which is not an equivalence unit
- principal_part_bound – an integer or None (default: None), a bound on the length of the principal part, i.e., the section of negative slope, of the Newton polygon of $G$
- assume_squarefree – whether or not to assume that $G$ is squarefree (default: False)
- assume_equivalence_irreducible – whether or not to assume that $G$ is equivalence irreducible (default: False)
- report_degree_bounds_and_caches – whether or not to include internal state with the returned value (used by `macLaneApproximants()` to speed up sequential calls)
- coefficients – the coefficients of $G$ in the $\phi()$-adic expansion if known (default: None)
- valuations – the valuations of coefficients if known (default: None)
- check – whether to check that $G$ is a squarefree monic non-constant integral polynomial and not an equivalence unit (default: True)
- allow_equivalent_key – whether to return valuations which end in essentially the same key polynomial as this valuation but have a higher valuation assigned to that key polynomial (default: True)

**EXAMPLES:**

We can use this method to perform the individual steps of `macLaneApproximants()`:
Starting from the Gauss valuation, a MacLane step branches off with some linear key polynomials in the above example:

```python
sage: v0 = GaussValuation(R, v)
sage: V1 = sorted(v0.mac_lane_step(f)); V1
[[ Gauss valuation induced by 2-adic valuation, v(x) = 2/5 ],
 [ Gauss valuation induced by 2-adic valuation, v(x) = 3 ],
 [ Gauss valuation induced by 2-adic valuation, v(x) = 11/9 ],
 [ Gauss valuation induced by 2-adic valuation, v(x) = 19/12 ],
 [ Gauss valuation induced by 2-adic valuation, v(x) = 5/6 ],
 [ Gauss valuation induced by 2-adic valuation, v(x) = 3/2 ]]
```
>>> from sage.all import *
>>> v0 = GaussValuation(R, v)
>>> V1 = sorted(v0.mac_lane_step(f)); V1
˓→needs sage.geometry.polyhedron
[[ Gauss valuation induced by 2-adic valuation, v(x) = 2/5 ],
 [ Gauss valuation induced by 2-adic valuation, v(x) = 3/5 ],
 [ Gauss valuation induced by 2-adic valuation, v(x) = 11/9 ],
 [ Gauss valuation induced by 2-adic valuation, v(x) = 3 ]]

The computation of MacLane approximants would now perform a MacLane step on each of these branches, note however, that a direct call to this method might produce some unexpected results:

```
sage: V1[1].mac_lane_step(f)  
˓→needs sage.geometry.polyhedron
[[ Gauss valuation induced by 2-adic valuation, v(x) = 3/5, v(x^5 + 8) = 5 ],
 [ Gauss valuation induced by 2-adic valuation, v(x) = 3/5, v(x^10 + 8*x^5 + ˓→64) = 7 ],
 [ Gauss valuation induced by 2-adic valuation, v(x) = 3 ],
 [ Gauss valuation induced by 2-adic valuation, v(x) = 11/9 ]]
```

```
>>> from sage.all import *
>>> V1[Integer(1)].mac_lane_step(f)  
˓→# needs sage.geometry.polyhedron
[[ Gauss valuation induced by 2-adic valuation, v(x) = 3/5, v(x^5 + 8) = 5 ],
 [ Gauss valuation induced by 2-adic valuation, v(x) = 3/5, v(x^10 + 8*x^5 + ˓→64) = 7 ],
 [ Gauss valuation induced by 2-adic valuation, v(x) = 3 ],
 [ Gauss valuation induced by 2-adic valuation, v(x) = 11/9 ]]
```

Note how this detected the two augmentations of V1[1] but also two other valuations that we had seen in the previous step and that are greater than V1[1]. To ignore such trivial augmentations, we can set allow_equivalent_key:

```
sage: V1[1].mac_lane_step(f, allow_equivalent_key=False)  
˓→needs sage.geometry.polyhedron
[[ Gauss valuation induced by 2-adic valuation, v(x) = 3/5, v(x^5 + 8) = 5 ],
 [ Gauss valuation induced by 2-adic valuation, v(x) = 3/5, v(x^10 + 8*x^5 + ˓→64) = 7 ]]  
```

```
>>> from sage.all import *
>>> V1[Integer(1)].mac_lane_step(f, allow_equivalent_key=False)  
˓→# needs sage.geometry.polyhedron
[[ Gauss valuation induced by 2-adic valuation, v(x) = 3/5, v(x^5 + 8) = 5 ],
 [ Gauss valuation induced by 2-adic valuation, v(x) = 3/5, v(x^10 + 8*x^5 + ˓→64) = 7 ]]  
```

minimal_representative(f)

Return a minimal representative for f, i.e., a pair e, a such that f is_equivalent() to ea, e is an equivalence unit, and a is_minimal() and monic.

INPUT:

- f – a non-zero polynomial which is not an equivalence unit

OUTPUT:

A factorization which has e as its unit and a as its unique factor.
ALGORITHM:

We use the algorithm described in the proof of Lemma 4.1 of [Mac1936II]. In the expansion $f = \sum_i f_i \phi^i$ take $e = f_i$ for the largest $i$ with $f_i \phi^i$ minimal (see `effective_degree()`). Let $h$ be the `equivalence_reciprocal()` of $e$ and take $a$ given by the terms of minimal valuation in the expansion of $ef$.

EXAMPLES:

```python
sage: # needs sage.libs.ntl
sage: R.<u> = Qq(4,10)
sage: S.<x> = R[]
sage: v = GaussValuation(S)
sage: v.minimal_representative(x + 2)
(1 + O(2^10))*x

sage: # needs sage.libs.ntl
sage: v = v.augmentation(x, 1)
sage: v.minimal_representative(x + 2)
(1 + O(2^10))*x + 2 + O(2^11)
sage: f = x^3 + 6*x + 4
sage: F = v.minimal_representative(f); F
(2 + 2^2 + O(2^11)) * ((1 + O(2^10))*x + 2 + O(2^11))
sage: v.is_minimal(F[0][0])
True
sage: v.is_equivalent(F.prod(), f)
True
```

```python
>>> from sage.all import *
>>> # needs sage.libs.ntl
>>> R = Qq(Integer(4),Integer(10), names=('u',)); (u,) = R._first_ngens(1)
>>> S = R['x']; (x,) = S._first_ngens(1)
>>> v = GaussValuation(S)
>>> v.minimal_representative(x + Integer(2))
(1 + O(2^10))*x

>>> # needs sage.libs.ntl
>>> v = v.augmentation(x, Integer(1))
>>> v.minimal_representative(x + Integer(2))
(1 + O(2^10))*x + 2 + O(2^11)
>>> f = x**Integer(3) + Integer(6)*x + Integer(4)
>>> F = v.minimal_representative(f); F
(2 + 2^2 + O(2^11)) * ((1 + O(2^10))*x + 2 + O(2^11))
>>> v.is_minimal(F[Integer(0)][Integer(0)])
True
>>> v.is_equivalent(F.prod(), f)
True
```
5.8 Augmented valuations on polynomial rings

Implements augmentations of (inductive) valuations.

AUTHORS:


EXAMPLES:

Starting from a Gauss valuation, we can create augmented valuations on polynomial rings:

```python
sage: R.<x> = QQ[]
sage: v = GaussValuation(R, QQ.valuation(2))
sage: w = v.augmentation(x, 1); w
[ Gauss valuation induced by 2-adic valuation, v(x) = 1 ]
sage: w(x)
1
```

This also works for polynomial rings over base rings which are not fields. However, much of the functionality is only available over fields:

```python
sage: R.<x> = ZZ[]
sage: v = GaussValuation(R, ZZ.valuation(2))
sage: w = v.augmentation(x, 1); w
[ Gauss valuation induced by 2-adic valuation, v(x) = 1 ]
sage: w(x)
1
```

REFERENCES:

Augmentations are described originally in [Mac1936I] and [Mac1936II]. An overview can also be found in Chapter 4 of [Rüt2014].

```python
class sage.rings.valuation.augmented_valuation.AugmentedValuationFactory
    Bases: UniqueFactory

    Factory for augmented valuations.
```

EXAMPLES:

This factory is not meant to be called directly. Instead, `augmentation()` of a valuation should be called:
Discrete Valuations and Discrete Pseudo-Valuations, Release 10.4

```python
sage: R.<x> = QQ[]
sage: v = GaussValuation(R, QQ.valuation(2))
sage: w = v.augmentation(x, 1)  # indirect doctest
```

```python
>>> from sage.all import *
>>> R = QQ['x']; (x,) = R._first_ngens(1)
>>> v = GaussValuation(R, QQ.valuation(Integer(2)))
>>> w = v.augmentation(x, Integer(1))  # indirect doctest
```

Note that trivial parts of the augmented valuation might be dropped, so you should not rely on `_base_valuation` to be the valuation you started with:

```python
sage: ww = w.augmentation(x, 2)
sage: ww._base_valuation is v
True
```

```python
>>> from sage.all import *
>>> ww = w.augmentation(x, Integer(2))
>>> ww._base_valuation is v
True
```

`create_key` (*base_valuation*, *phi*, *mu*, *check=True*)

Create a key which uniquely identifies the valuation over `base_valuation` which sends `phi` to `mu`.

**Note:** The uniqueness that this factory provides is not why we chose to use a factory. However, it makes pickling and equality checks much easier. At the same time, going through a factory makes it easier to enforce that all instances correctly inherit methods from the parent Hom space.

`create_object` (*version*, *key*)

Create the augmented valuation represented by `key`.

**class** `sage.rings.valuation.augmented_valuation.AugmentedValuation_base` (*parent*, *v*, *phi*, *mu*)

**Bases:** `InductiveValuation`

An augmented valuation is a discrete valuation on a polynomial ring. It extends another discrete valuation `v` by setting the valuation of a polynomial `f` to the minimum of `v(f^i)\mu^i` when writing `f = \sum_i f_i \phi^i`.

**INPUT:**

- `v` – a `InductiveValuation` on a polynomial ring
- `phi` – a `key polynomial` over `v`
- `mu` – a rational number such that `mu > v(phi)` or `infinity`

**EXAMPLES:**

```python
sage: # needs sage.rings.number_field
sage: K.<u> = CyclotomicField(5)
sage: R.<x> = K[]
sage: v = GaussValuation(R, K.valuation(2))
sage: w = v.augmentation(x, 1/2); w  # indirect doctest
[ Gauss valuation induced by 2-adic valuation, v(x) = 1/2 ]
sage: ww = w.augmentation(x^4 + 2*x^2 + 4*u, 3); ww
[ Gauss valuation induced by 2-adic valuation, v(x) = 1/2, v(x^4 + 2*x^2 + 4*u) = 3 ]
```

5.8. Augmented valuations on polynomial rings 83
>>> from sage.all import *

# needs sage.rings.number_field

>>>

K = CyclotomicField(Integer(5), names=('u',)); (u,) = K._first_ngens(1)

>>>

R = K['x']; (x,) = R._first_ngens(1)

>>>

v = GaussValuation(R, K.valuation(Integer(2)))

>>>

w = v.augmentation(x, Integer(1)/Integer(2)); w  # indirect doctest
[ Gauss valuation induced by 2-adic valuation, v(x) = 1/2 ]

>>>

ww = w.augmentation(x**Integer(4) + Integer(2)*x**Integer(2) + Integer(4)*u, →Integer(3)); ww
[ Gauss valuation induced by 2-adic valuation, v(x^4 + 2*x^2 + 4*u) = 3 ]

E()

Return the ramification index of this valuation over its underlying Gauss valuation.

EXAMPLES:

sage: # needs sage.libs.ntl
sage: R.<u> = Qq(4, 5)
sage: S.<x> = R[]
sage: v = GaussValuation(S)
sage: w = v.augmentation(x^2 + x + u, 1)
sage: w.E() 1
sage: w = v.augmentation(x, 1/2)
sage: w.E() 2

F()

Return the degree of the residue field extension of this valuation over the underlying Gauss valuation.

EXAMPLES:

sage: # needs sage.libs.ntl
sage: R = Qq(Integer(4), Integer(5), names=('u',)); (u,) = R._first_ngens(1)
sage: S = R['x']; (x,) = S._first_ngens(1)
sage: v = GaussValuation(S)
sage: w = v.augmentation(x**Integer(2) + x + u, Integer(1))
sage: w.E() 1
sage: w = v.augmentation(x, Integer(1)/Integer(2))

sage: w.E() 2

84 Chapter 5. More Details
Discrete Valuations and Discrete Pseudo-Valuations, Release 10.4

```python
>>> from sage.all import *
>>> # needs sage.libsntl

>>> R = QQ(Integer(4), Integer(5), names=('u',)); (u,) = R._first_ngens(1)
>>> S = R['x']; (x,) = S._first_ngens(1)
>>> v = GaussValuation(S)
>>> w = v.augmentation(x**Integer(2) + x + u, Integer(1))
>>> w.F()
2
>>> w = v.augmentation(x, Integer(1)/Integer(2))
>>> w.F()
1
```

`augmentation_chain()`

Return a list with the chain of augmentations down to the underlying Gauss valuation.

**EXAMPLES:**

```python
sage: R.<x> = QQ[]
sage: v = GaussValuation(R, QQ.valuation(2))
sage: w = v.augmentation(x, 1)
sage: w.augmentation_chain()
[[ Gauss valuation induced by 2-adic valuation, v(x) = 1 ],
Gauss valuation induced by 2-adic valuation]
```

For performance reasons, (and to simplify the underlying implementation,) trivial augmentations might get dropped. You should not rely on `augmentation_chain()` to contain all the steps that you specified to create the current valuation:

```python
sage: ww = w.augmentation(x, 2)
sage: ww.augmentation_chain()
[[ Gauss valuation induced by 2-adic valuation, v(x) = 2 ],
Gauss valuation induced by 2-adic valuation]
```

`change_domain(ring)`

Return this valuation over `ring`.

**EXAMPLES:**

We can change the domain of an augmented valuation even if there is no coercion between rings:

```python
sage: # needs sage.rings.number_field
sage: R.<x> = GaussianIntegers()
```

(continues on next page)
element_with_valuation(s)

Create an element of minimal degree and of valuation s.

INPUT:

• s – a rational number in the value group of this valuation

OUTPUT:

An element in the domain of this valuation

EXAMPLES:

sage: # needs sage.libsntl
sage: R.<u> = Qq(4, 5)
sage: S.<x> = R[]
sage: v = GaussValuation(R, GaussianIntegers().valuation(Integer(2)))
sage: w = v.augmentation(x^2 + x + u, 1/2)
sage: w.element_with_valuation(0)
1 + O(2^5)
sage: w.element_with_valuation(1/2)
(1 + O(2^5))*x^2 + (1 + O(2^5))*x + u + O(2^5)
sage: w.element_with_valuation(1)
2 + O(2^6)
sage: c = w.element_with_valuation(-1/2); c
(2^-1 + O(2^4))*x^2 + (2^-1 + O(2^4))*x + u*2^-1 + O(2^4)
sage: w(c)
-1/2
sage: w.element_with_valuation(1/3)
Traceback (most recent call last):
... ValueError: s must be in the value group of the valuation but 1/3 is not in Additive Abelian Group generated by 1/2.

sage: # needs sage.libsntl
sage: R = Qq(Integer(4), Integer(5), names=('u',)); (u,) = R._first_ngens(1)
sage: S = R['x']; (x,) = S._first_ngens(1)
sage: v = GaussValuation(S)
sage: w = v.augmentation(x**Integer(2) + x + u, Integer(1)/Integer(2))
sage: w.element_with_valuation(Integer(0))
1 + O(2^5)
sage: w.element_with_valuation(Integer(1)/Integer(2))
(1 + O(2^5))*x^2 + (1 + O(2^5))*x + u + O(2^5)
sage: w.element_with_valuation(Integer(1))
Discrete Valuations and Discrete Pseudo-Valuations, Release 10.4

(continued from previous page)

```python
2 + O(2^6)
>> c = w.element_with_valuation(-Integer(1)/Integer(2)); c
(2^-1 + O(2^4))*x^2 + (2^-1 + O(2^4))*x + u*2^-1 + O(2^4)
>> w(c)
-1/2
>> w.element_with_valuation(Integer(1)/Integer(3))
Traceback (most recent call last):
  ... ValueError: s must be in the value group of the valuation
but 1/3 is not in Additive Abelian Group generated by 1/2.
```

**equivalence_unit** *(s, reciprocal=False)*

Return an equivalence unit of minimal degree and valuation s.

**INPUT:**

- s – a rational number
- reciprocal – a boolean (default: False); whether or not to return the equivalence unit as the `equivalence_reciprocal()` of the equivalence unit of valuation −s.

**OUTPUT:**

A polynomial in the domain of this valuation which `is_equivalence_unit()` for this valuation.

**EXAMPLES:**

```python
sage: # needs sage.libsntl
sage: R.<u> = Qq(4, 5)
sage: S.<x> = R[
]
sage: v = GaussValuation(S)
sage: w = v.augmentation(x^2 + x + u, 1)
sage: w.equivalence_unit(0)
1 + O(2^5)
sage: w.equivalence_unit(-4)
2^-4 + O(2)
```

```python
>>> from sage.all import *
```

```python
sage: R = Qq(Integer(4), Integer(5), names=('u',)); (u,) = R._first_ngens(1)
sage: S = R['x']; (x,) = S._first_ngens(1)
sage: v = GaussValuation(S)
sage: w = v.augmentation(x^2 + x + u, 1)
sage: w.equivalence_unit(Integer(0))
1 + O(2^5)
sage: w.equivalence_unit(-Integer(4))
2^-4 + O(2)
```

Since an equivalence unit is of effective degree zero, $\phi$ must not divide it. Therefore, its valuation is in the value group of the base valuation:

```python
sage: w = v.augmentation(x, 1/2)  #...
←needs sage.libsntl
sage: w.equivalence_unit(3/2)  #...
←needs sage.libsntl
Traceback (most recent call last):
  ...
ValueError: 3/2 is not in the value semigroup of 2-adic valuation
```

(continues on next page)
An equivalence unit might not be integral, even if \( s \geq 0 \):

```sage
sage: w = v.augmentation(x, 3/4)  # needs sage.libs.ntl
sage: ww = w.augmentation(x^4 + 8, 5)  # needs sage.libs.ntl
sage: ww.equivalence_unit(1/2)  # needs sage.libs.ntl
(2^-1 + O(2^4))*x^2
```

```
>>> from sage.all import *

>>> w = v.augmentation(x, Integer(3)/Integer(4))  # needs sage.libs.ntl
>>> ww = w.augmentation(x^4 + Integer(8), Integer(5))  # needs sage.libs.ntl
>>> ww.equivalence_unit(Integer(1)/Integer(2))  # needs sage.libs.ntl
(2^-1 + O(2^4))*x^2
```

**extensions** *(ring)*

Return the extensions of this valuation to \( \text{ring} \).

**EXAMPLES:**

```sage
sage: R.<x> = QQ[]
sage: v = GaussValuation(R, QQ.valuation(Integer(2)))
sage: w = v.augmentation(x^2 + x + 1, 1)
sage: w.extensions(GaussianIntegers().fraction_field()[:x:]
[[ Gauss valuation induced by 2-adic valuation, v(x^2 + x + 1) = 1 ]]
```

```sage
>>> from sage.all import *

>>> R = QQ['x']; (x,) = R._first_ngens(1)
>>> v = GaussValuation(R, QQ.valuation(Integer(2)))
>>> w = v.augmentation(x**Integer(2) + x + Integer(1), Integer(1))
>>> w.extensions(GaussianIntegers().fraction_field()[:x:]
[[ Gauss valuation induced by 2-adic valuation, v(x^2 + x + 1) = 1 ]]
```

**is_gauss_valuation()**
Return whether this valuation is a Gauss valuation.

EXAMPLES:

```python
sage: R.<x> = QQ[]
sage: v = GaussValuation(R, QQ.valuation(2))
sage: w = v.augmentation(x^2 + x + 1, 1)
sage: w.is_gauss_valuation()
False
```

```python
>>> from sage.all import *
>>> R = QQ['x']; (x,) = R._first_ngens(1)
>>> v = GaussValuation(R, QQ.valuation(Integer(2)))
>>> w = v.augmentation(x**Integer(2) + x + Integer(1), Integer(1))

>>> w.is_gauss_valuation()
False
```

`is_negative_pseudo_valuation()`

Return whether this valuation attains $-\infty$.

EXAMPLES:

No element in the domain of an augmented valuation can have valuation $-\infty$, so this method always returns False:

```python
sage: R.<x> = QQ[]
sage: v = GaussValuation(R, valuations.TrivialValuation(QQ))
sage: w = v.augmentation(x, infinity)
sage: w.is_negative_pseudo_valuation()
False
```

```python
>>> from sage.all import *
>>> R = QQ['x']; (x,) = R._first_ngens(1)
>>> v = GaussValuation(R, valuations.TrivialValuation(QQ))
>>> w = v.augmentation(x, infinity)

>>> w.is_negative_pseudo_valuation()
False
```

`is_trivial()`

Return whether this valuation is trivial, i.e., zero outside of zero.

EXAMPLES:

```python
sage: R.<x> = QQ[]
sage: v = GaussValuation(R, QQ.valuation(2))
sage: w = v.augmentation(x^2 + x + 1, 1)
sage: w.is_trivial()
False
```

```python
>>> from sage.all import *
>>> R = QQ['x']; (x,) = R._first_ngens(1)
>>> v = GaussValuation(R, QQ.valuation(Integer(2)))
>>> w = v.augmentation(x**Integer(2) + x + Integer(1), Integer(1))

>>> w.is_trivial()
False
```
**monic_integral_model** \((G)\)

Return a monic integral irreducible polynomial which defines the same extension of the base ring of the domain as the irreducible polynomial \(G\) together with maps between the old and the new polynomial.

**EXAMPLES:**

```sage
sage: R.<x> = QQ[]
sage: v = GaussValuation(R, QQ.valuation(2))
sage: w = v.augmentation(x^2 + x + 1, 1)
sage: w.monic_integral_model(5*x^2 + 1/2*x + 1/4)
(Ring endomorphism of Univariate Polynomial Ring in x over Rational Field
 Defn: x |--> 1/2*x,
Ring endomorphism of Univariate Polynomial Ring in x over Rational Field
 Defn: x |--> 2*x,
x^2 + 1/5*x + 1/5)
```

**psi()**

Return the minimal polynomial of the residue field extension of this valuation.

**OUTPUT:**

A polynomial in the residue ring of the base valuation

**EXAMPLES:**

```sage
sage: # needs sage.libsntl
sage: R.<u> = Qq(4, 5)
sage: S.<x> = R[]
sage: v = GaussValuation(S)
sage: w = v.augmentation(x^2 + x + u, 1/2)
sage: w.psi()
x^2 + x + u0
sage: ww = w.augmentation((x^2 + x + u)^2 + 2, 5/3)
sage: ww.psi()
x + 1
```

(continues on next page)
restriction (ring)

Return the restriction of this valuation to \(\text{ring}\).

EXAMPLES:

```python
sage: # needs sage.rings.number_field
sage: K = GaussianIntegers().fraction_field()
sage: R.<x> = K[]
sage: v = GaussValuation(R, K.valuation(2))
sage: w = v.augmentation(x^2 + x + 1, 1)
```

```python
w.restriction(QQ[x])
```

```
# needs sage.libs.singular
```

Gauss valuation induced by 2-adic valuation, \(v(x^2 + x + 1) = 1\)

```python
from sage.all import *
```

```python
R = QQ[x]; (x,) = R._first_ngens(1)
```

```python
v = GaussValuation(R, QQ.valuation(Integer(2)))
```

```python
w = v.augmentation(x**Integer(2) + x + Integer(1), Integer(1))
```

```python
w.restriction(QQ['x'])
```

```
# needs sage.libs.singular
```

Gauss valuation induced by 2-adic valuation, \(v(x^2 + x + 1) = 1\)

scale (scalar)

Return this valuation scaled by \(\text{scalar}\).

EXAMPLES:

```python
sage: R.<x> = QQ[]
```

```python
v = GaussValuation(R, QQ.valuation(2))
```

```python
w = v.augmentation(x^2 + x + 1, 1)
```

```python
3*w
```

```
# indirect doctest
```

Gauss valuation induced by 3 * 2-adic valuation, \(v(x^2 + x + 1) = 3\)

```python
from sage.all import *
```

```python
R = QQ['x']; (x,) = R._first_ngens(1)
```

```python
v = GaussValuation(R, QQ.valuation(Integer(2)))
```

```python
w = v.augmentation(x**Integer(2) + x + Integer(1), Integer(1))
```

```python
Integer(3)*w
```

```
# indirect doctest
```

Gauss valuation induced by 3 * 2-adic valuation, \(v(x^2 + x + 1) = 3\)

uniformizer ()

Return a uniformizing element for this valuation.

EXAMPLES:

```python
sage: R.<x> = QQ[]
```

```python
v = GaussValuation(R, QQ.valuation(2))
```

```python
w = v.augmentation(x^2 + x + 1, 1)
```

```python
w.uniformizer()
```

2
class sage.rings.valuation.augmented_valuation.FinalAugmentedValuation(parent, v, phi, mu)

Bases: AugmentedValuation_base, FinalInductiveValuation

An augmented valuation which can not be augmented anymore, either because it augments a trivial valuation or because it is infinite.

EXAMPLES:

```
sage: R.<x> = QQ[]
sage: v = GaussValuation(R, valuations.TrivialValuation(QQ))
sage: w = v.augmentation(x, 1)
```

**lift**(F)

Return a polynomial which reduces to F.

**INPUT:**

- F – an element of the `residue_ring()`

**ALGORITHM:**

We simply undo the steps performed in `reduce()`.

**OUTPUT:**

A polynomial in the domain of the valuation with reduction F

**EXAMPLES:**

```
sage: R.<x> = QQ[]
sage: v = GaussValuation(R, valuations.TrivialValuation(QQ))

sage: w = v.augmentation(x, 1)
sage: w.lift(1/2)
1/2

sage: w = v.augmentation(x^2 + x + 1, infinity)
sage: w.lift(w.residue_ring().gen())
# needs sage.rings.number_field
x
```

(continues on next page)
A case with non-trivial base valuation:

```python
sage: # needs sage.libs.ntl
sage: R.<u> = Qq(4, 10)
sage: S.<x> = R[]
sage: v = GaussValuation(S)
sage: w = v.augmentation(x^2 + x + u, infinity)
sage: w.lift(w.residue_ring().gen())
```

reduce \((f, \text{check}=True, \text{degree\_bound}=\text{None}, \text{coefficients}=\text{None}, \text{valuations}=\text{None})\)

Reduce \(f\) modulo this valuation.

**INPUT:**

- \(f\) – an element in the domain of this valuation
- \(\text{check}\) – whether or not to check whether \(f\) has non-negative valuation (default: True)
- \(\text{degree\_bound}\) – an a-priori known bound on the degree of the result which can speed up the computation (default: not set)
- \(\text{coefficients}\) – the coefficients of \(f\) as produced by \(\text{coefficients}()\) or None (default: None); this can be used to speed up the computation when the expansion of \(f\) is already known from a previous computation.
- \(\text{valuations}\) – the valuations of \(\text{coefficients}\) or None (default: None); ignored

**OUTPUT:**

an element of the \(\text{residue\_ring()}\) of this valuation, the reduction modulo the ideal of elements of positive valuation

**EXAMPLES:**

```python
sage: R.<x> = QQ[]
sage: v = GaussValuation(R, valuations.TrivialValuation(QQ))
sage: w = v.augmentation(x, 1)
```
Discrete Valuations and Discrete Pseudo-Valuations, Release 10.4

sage: w.reduce(x^2 + x + 1)
1

sage: w = v.augmentation(x^2 + x + 1, infinity)
sage: w.reduce(x)  # needs sage.rings.number_field
u1

residue_ring()

Return the residue ring of this valuation, i.e., the elements of non-negative valuation modulo the elements of positive valuation.

EXAMPLES:

sage: R.<x> = QQ[]
sage: v = GaussValuation(R, valuations.TrivialValuation(QQ))
sage: w = v.augmentation(x, 1)
sage: w.residue_ring()
Rational Field
sage: w = v.augmentation(x^2 + x + 1, infinity)
sage: w.residue_ring()  # needs sage.rings.number_field
Number Field in u1 with defining polynomial x^2 + x + 1

An example with a non-trivial base valuation:

sage: v = GaussValuation(R, QQ.valuation(2))
sage: w = v.augmentation(x^2 + x + 1, infinity)
Discrete Valuations and Discrete Pseudo-Valuations, Release 10.4

(continued from previous page)

```python
sage: w.residue_ring()  # needs sage.rings.finite_rings
Finite Field in u1 of size 2^2
```

```python
>>> from sage.all import *

>>> v = GaussValuation(R, QQ.valuation(Integer(2)))
>>> w = v.augmentation(x**Integer(2) + x + Integer(1), infinity)
>>> w.residue_ring()  # needs sage.rings.finite_rings
Finite Field in u1 of size 2^2
```

Since trivial extensions of finite fields are not implemented, the resulting ring might be identical to the residue ring of the underlying valuation:

```python
sage: w = v.augmentation(x, infinity)
sage: w.residue_ring()
Finite Field of size 2
```

```python
>>> from sage.all import *

>>> R = QQ['x']; (x,) = R._first_ngens(1)
>>> v = GaussValuation(R, valuations.TrivialValuation(QQ))
>>> w = v.augmentation(x, Integer(1))

class sage.rings.valuation.augmented_valuation.FinalFiniteAugmentedValuation(parent, v, phi, mu)

Bases: FiniteAugmentedValuation, FinalAugmentedValuation

An augmented valuation which is discrete, i.e., which assigns a finite valuation to its last key polynomial, but which can not be further augmented.

EXAMPLES:

```python
sage: R.<x> = QQ[]
sage: v = GaussValuation(R, valuations.TrivialValuation(QQ))
sage: w = v.augmentation(x, 1)
```

```python
>>> from sage.all import *

>>> R = QQ['x']; (x,) = R._first_ngens(1)
>>> v = GaussValuation(R, valuations.TrivialValuation(QQ))
>>> w = v.augmentation(x, Integer(1))
```

class sage.rings.valuation.augmented_valuation.FiniteAugmentedValuation(parent, v, phi, mu)

Bases: AugmentedValuation_base, FiniteInductiveValuation

A finite augmented valuation, i.e., an augmented valuation which is discrete, or equivalently an augmented valuation which assigns to its last key polynomial a finite valuation.

EXAMPLES:

```python
sage: # needs sage.libs.ntl
sage: R.<u> = Qq(4, 5)
```
Discrete Valuations and Discrete Pseudo-Valuations, Release 10.4

\begin{verbatim}
sage: S.<x> = R[]
sage: v = GaussValuation(S)
sage: w = v.augmentation(x^2 + x + u, 1/2)

>>> from sage.all import *
>>> # needs sage.libs.ntl
>>> R = Qq(Integer(4), Integer(5), names=('u',)); (u,) = R._first_ngens(1)
>>> S = R['x']; (x,) = S._first_ngens(1)
>>> v = GaussValuation(S)
>>> w = v.augmentation(x^2 + x + u, Integer(1)/Integer(2))

lower_bound(f)

Return a lower bound of this valuation at \( f \).

Use this method to get an approximation of the valuation of \( f \) when speed is more important than accuracy.

ALGORITHM:

The main cost of evaluation is the computation of the \texttt{coefficients()} of the \texttt{phi()}-adic expansion of \( f \) (which often leads to coefficient bloat.) So unless \texttt{phi()} is trivial, we fall back to valuation which this valuation augments since it is guaranteed to be smaller everywhere.

EXAMPLES:

\begin{verbatim}
sage: # needs sage.libs.ntl
sage: R.<u> = Qq(4, 5)
sage: S.<x> = R[]
sage: v = GaussValuation(S)
sage: w = v.augmentation(x^2 + x + u, 1/2)
sage: w.lower_bound(x^2 + x + u)
0
\end{verbatim}

\end{verbatim}

simplify (f, error=None, force=False, effective_degree=None, size_heuristic_bound=32, phiadic=False)

Return a simplified version of \( f \).

Produce an element which differs from \( f \) by an element of valuation strictly greater than the valuation of \( f \) (or strictly greater than \texttt{error} if set.)

INPUT:

- \( f \) – an element in the domain of this valuation
- \texttt{error} – a rational, infinity, or \texttt{None} (default: \texttt{None}), the error allowed to introduce through the simplification
- \texttt{force} – whether or not to simplify \( f \) even if there is heuristically no change in the coefficient size of \( f \) expected (default: \texttt{False})
- \texttt{effective_degree} – when set, assume that coefficients beyond \texttt{effective_degree} in the \texttt{phi()}-adic development can be safely dropped (default: \texttt{None})
• `size_heuristic_bound` – when `force` is not set, the expected factor by which the coefficients need to shrink to perform an actual simplification (default: 32)

• `phiadic` – whether to simplify the coefficients in the \(\phi\)-adic expansion recursively. This often times leads to huge coefficients in the \(x\)-adic expansion (default: `False`, i.e., use an \(x\)-adic expansion.)

EXAMPLES:

```python
sage: # needs sage.libs.ntl
sage: R.<u> = Qq(4, 5)
sage: S.<x> = R[]
sage: v = GaussValuation(S)
sage: w = v.augmentation(x^2 + x + u, 1/2)
sage: w.simplify(x^10/2 + 1, force=True)
(u + 1)*2^-1 + O(2^4)
```

Check that Issue #25607 has been resolved, i.e., the coefficients in the following example are small:

```python
sage: # needs sage.libs.ntl sage.rings.number_field
sage: R.<x> = QQ[]
sage: K.<a> = NumberField(x^3 + 6)
sage: R.<x> = K[]
sage: v = GaussValuation(R, K.valuation(2))
sage: v = v.augmentation(x, 3/2)
sage: v = v.augmentation(x^2 + 8, 13/4)
sage: v = v.augmentation(x^4 + 16*x^2 + 32*x + 64, 20/3)
sage: F.<x> = FunctionField(K)
sage: S.<y> = F[]
sage: v = F.valuation(v)
sage: G = y^2 - 2*x^5 + 8*x^3 + 80*x^2 + 128*x + 192
sage: v.mac_lane_approximants(G)
[[ Gauss valuation induced by Valuation on rational function field induced by
  [ Gauss valuation induced by 2-adic valuation, v(x) = 3/2,
    v(x^2 + 8) = 13/4, v(x^4 + 16*x^2 + 32*x + 64) = 20/3 ],
  v(y + 4*x + 8) = 31/8 ]]
```

(continues on next page)
upper_bound\( (f) \)

Return an upper bound of this valuation at \( f \).

Use this method to get an approximation of the valuation of \( f \) when speed is more important than accuracy.

**ALGORITHM:**

Any entry of \( \text{valuations()} \) serves as an upper bound. However, computation of the \( \text{phi()} \)-adic expansion of \( f \) is quite costly. Therefore, we produce an upper bound on the last entry of \( \text{valuations()} \), namely the valuation of the leading coefficient of \( f \) plus the valuation of the appropriate power of \( \text{phi()} \).

**EXAMPLES:**

```python
sage: from sage.all import *
>>> from sage.all import *  
```

valuations \( (f, \text{coefficients}=\text{None, call_error=}\text{False}) \)

Return the valuations of the \( f_i \phi^i \) in the expansion \( f = \sum_i f_i \phi^i \).

**INPUT:**

- \( f \) – a polynomial in the domain of this valuation
- \( \text{coefficients} \) – the coefficients of \( f \) as produced by \( \text{coefficients()} \) or \( \text{None} \) (default: \( \text{None} \)); this can be used to speed up the computation when the expansion of \( f \) is already known from a previous computation.
- \( \text{call_error} \) – whether or not to speed up the computation by assuming that the result is only used to compute the valuation of \( f \) (default: \( \text{False} \))

**OUTPUT:**

An iterator over rational numbers (or infinity) \([v(f_0), v(f_1), \ldots]\)

**EXAMPLES:**

```python
sage: # needs sage.libsntl
sage: R.<u> = QQ(4, 5)
sage: v = GaussValuation(S)
sage: w = v.augmentation(x^2 + x + u, 1/2)
sage: w.upper_bound(x^2 + x + u)
1/2
```
Discrete Valuations and Discrete Pseudo-Valuations, Release 10.4

```sage
# needs sage.libs.ntl
R.<u> = Qq(4, 5)
S.<x> = R[]
v = GaussValuation(S)
w = v.augmentation(x^2 + x + u, 1/2)
list(w.valuations( x^2 + 1 ))
[0, 1/2]
ww = w.augmentation((x^2 + x + u)^2 + 2, 5/3)
list(ww.valuations( ((x^2 + x + u)^2 + 2)^3 ))
[+Infinity, +Infinity, +Infinity, 5]

from sage.all import *
R = Qq(Integer(4), Integer( 5), names=(u,)); (u,) = R._first_ngens(1)
S = R['x']; (x,) = S._first_ngens(1)
v = GaussValuation(S)
w = v.augmentation(x**Integer(2) + x + u, Integer(1)/Integer(2))
list(w.valuations( x**Integer(2) + Integer(1) ))
[0, 1/2]
ww = w.augmentation((x**Integer(2) + x + u)**Integer(2) + Integer(2), Integer(5)/Integer(3))
list(ww.valuations( ((x**Integer(2) + x + u)**Integer(2) + Integer(2))**Integer(3) ))
[+Infinity, +Infinity, +Infinity, 5]
```

```sage
value_group()

Return the value group of this valuation.

EXAMPLES:

```sage
# needs sage.libs.ntl
R.<u> = Qq(4, 5)
S.<x> = R[]
v = GaussValuation(S)
w = v.augmentation(x^2 + x + u, 1/2)
w.value_group()
Additive Abelian Group generated by 1/2
ww = w.augmentation((x^2 + x + u)^2 + 2, 5/3)
ww.value_group()
Additive Abelian Group generated by 1/6
```

```sage
value_semigroup()

Return the value semigroup of this valuation.

EXAMPLES:

```sage
# needs sage.libs.ntl
R = Qq(Integer(4), Integer( 5), names=('u',)); (u,) = R._first_ngens(1)
S = R['x']; (x,) = S._first_ngens(1)
v = GaussValuation(S)
w = v.augmentation(x**Integer(2) + x + u, Integer(1)/Integer(2))
w.value_group()
Additive Abelian Group generated by 1/2
ww = w.augmentation((x**Integer(2) + x + u)**Integer(2) + Integer(2), Integer(5)/Integer(3))
ww.value_group()
Additive Abelian Group generated by 1/6
```

5.8. Augmented valuations on polynomial rings
Discrete Valuations and Discrete Pseudo-Valuations, Release 10.4

```python
sage: # needs sage.libs.ntl
sage: R.<u> = Zq(4, 5)
sage: S.<x> = R[]
sage: v = GaussValuation(S)
sage: w = v.augmentation(x^2 + x + u, 1/2)
sage: w.value_semigroup()
Additive Abelian Semigroup generated by 1/2
sage: ww = w.augmentation((x^2 + x + u)^2 + 2, 5/3)
sage: ww.value_semigroup()
Additive Abelian Semigroup generated by 1/2, 5/3
```

```python
>>> from sage.all import *
>>> # needs sage.libs.ntl
>>> R = Zq(Integer(4), Integer(5), names=('u',)); (u,) = R._first_ngens(1)
>>> S = R['x']; (x,) = S._first_ngens(1)
>>> v = GaussValuation(S)
>>> w = v.augmentation(x**Integer(2) + x + u, Integer(1)/Integer(2))
>>> w.value_semigroup()
Additive Abelian Semigroup generated by 1/2
>>> ww = w.augmentation((x**Integer(2) + x + u)**Integer(2) + Integer(2),
                        Integer(5)/Integer(3))
>>> ww.value_semigroup()
Additive Abelian Semigroup generated by 1/2, 5/3
```

class sage.rings.valuation.augmented_valuation.InfiniteAugmentedValuation

Bases: FinalAugmentedValuation, InfiniteInductiveValuation

An augmented valuation which is infinite, i.e., which assigns valuation infinity to its last key polynomial (and which can therefore not be augmented further.)

EXAMPLES:

```
sage: # needs sage.libs.ntl
sage: R.<x> = QQ[]
sage: v = GaussValuation(R, QQ.valuation(2))
sage: w = v.augmentation(x, infinity)
```

lower_bound(f)

Return a lower bound of this valuation at f.

Use this method to get an approximation of the valuation of f when speed is more important than accuracy.

EXAMPLES:

```
sage: # needs sage.libs.ntl
sage: R.<u> = QQ(4, 5)
sage: S.<x> = R[]
sage: v = GaussValuation(S)
sage: w = v.augmentation(x^2 + x + u, infinity)
```

(continues on next page)
Discrete Valuations and Discrete Pseudo-Valuations, Release 10.4

sage: w.lower_bound(x^2 + x + u)
+Infinity

>>> from sage.all import *
>>> # needs sage.libs.ntl
>>> R = QQ(Integer(4), Integer(5), names=('u',)); (u,) = R._first_ngens(1)
>>> S = R['x']; (x,) = S._first_ngens(1)
>>> v = GaussValuation(S)
>>> w = v.augmentation(x**Integer(2) + x + u, infinity)
>>> w.lower_bound(x**Integer(2) + x + u)
+Infinity

**simplify** (*f*, *error=None*, *force=False*, *effective_degree=None*)

Return a simplified version of *f*.

Produce an element which differs from *f* by an element of valuation strictly greater than the valuation of *f* (or strictly greater than *error* if set.)

**INPUT:**

- *f* – an element in the domain of this valuation
- *error* – a rational, infinity, or None (default: None), the error allowed to introduce through the simplification
- *force* – whether or not to simplify *f* even if there is heuristically no change in the coefficient size of *f* expected (default: False)
- *effective_degree* – ignored; for compatibility with other simplify methods

**EXAMPLES:**

sage: # needs sage.libs.ntl
sage: R.<u> = QQ(4, 5)

sage: S.<x> = R[]

sage: v = GaussValuation(S)

sage: w = v.augmentation(x**Integer(2) + x + u, infinity)

sage: w.simplify(x**Integer(10)/Integer(2) + Integer(1), force=True)
(u + 1)*2^-1 + O(2^4)

upper_bound(*f*)

Return an upper bound of this valuation at *f*.

Use this method to get an approximation of the valuation of *f* when speed is more important than accuracy.

**EXAMPLES:**

sage: # needs sage.libs.ntl
sage: R.<u> = QQ(4, 5)

(continues on next page)
Discrete Valuations and Discrete Pseudo-Valuations, Release 10.4

Continued from previous page:

```python
sage: S.<x> = R[]
sage: v = GaussValuation(S)
sage: w = v.augmentation(x^2 + x + u, infinity)
sage: w.upper_bound(x^2 + x + u)
+Infinity
```

```python
>>> from sage.all import *
>>> # needs sage.libs.ntl
>>> R = Qq(Integer(4), Integer(5), names=('u',)); (u,) = R._first_ngens(1)
>>> S = R['x']; (x,) = S._first_ngens(1)
>>> v = GaussValuation(S)
>>> w = v.augmentation(x**Integer(2) + x + u, infinity)
>>> w.upper_bound(x**Integer(2) + x + u)
+Infinity
```

valuation($f$, coefficients=None, call_error=False)

Return the valuations of the $f_i\phi^i$ in the expansion $f = \sum_i f_i\phi^i$.

**INPUT:**

- $f$ – a polynomial in the domain of this valuation
- coefficients – the coefficients of $f$ as produced by coefficients() or None (default: None); this can be used to speed up the computation when the expansion of $f$ is already known from a previous computation.
- call_error – whether or not to speed up the computation by assuming that the result is only used to compute the valuation of $f$ (default: False)

**OUTPUT:**

An iterator over rational numbers (or infinity) $[v(f_0), v(f_1\phi), \ldots]$.

**EXAMPLES:**

```python
sage: # needs sage.libs.ntl
sage: R.<u> = Qq(4, 5)
sage: S.<x> = R[]
sage: v = GaussValuation(S)
sage: w = v.augmentation(x, infinity)
sage: list(w.valuations(x^2 + 1))
[0, +Infinity, +Infinity]
```

```python
>>> from sage.all import *
>>> # needs sage.libs.ntl
>>> R = Qq(Integer(4), Integer(5), names=('u',)); (u,) = R._first_ngens(1)
>>> S = R['x']; (x,) = S._first_ngens(1)
>>> v = GaussValuation(S)
>>> w = v.augmentation(x, infinity)
>>> list(w.valuations(x**Integer(2) + Integer(1)))
[0, +Infinity, +Infinity]
```

value_group()  

Return the value group of this valuation.

**EXAMPLES:**
Discrete Valuations and Discrete Pseudo-Valuations, Release 10.4

```python
sage: # needs sage.libs.ntl
sage: R.<u> = Qq(4, 5)
sage: S.<x> = R[]
sage: v = GaussValuation(S)
sage: w = v.augmentation(x, infinity)
sage: w.value_group()
Additive Abelian Group generated by 1
```

```python
>>> from sage.all import *
>>> # needs sage.libs.ntl
>>> R = Qq(Integer(4), Integer(5), names=('u',)); (u,) = R._first_ngens(1)
>>> S = R['x']; (x,) = S._first_ngens(1)
>>> v = GaussValuation(S)
>>> w = v.augmentation(x, infinity)
>>> w.value_group()
Additive Abelian Group generated by 1
```

```python
value_semigroup()
```

Return the value semigroup of this valuation.

**EXAMPLES:**

```python
sage: # needs sage.libs.ntl
sage: R.<x> = QQ[]
sage: v = GaussValuation(R, QQvaluation(2))
sage: w = v.augmentation(x^2 + x + 1, 1)
```

```python
>>> from sage.all import *
>>> R = QQ['x']; (x,) = R._first_ngens(1)
>>> v = GaussValuation(R, QQvaluation(Integer(2)))
>>> w = v.augmentation(x^*Integer(2) + x + Integer(1), Integer(1))
```

**class** sage.rings.valuation.augmented_valuation.NonFinalAugmentedValuation

Bases: AugmentedValuation_base, NonFinalInductiveValuation

An augmented valuation which can be augmented further.

**EXAMPLES:**

```python
```
\textbf{\texttt{lift}} \((F, \text{report\_coefficients}=\text{False})\)

Return a polynomial which reduces to \(F\).

\textbf{INPUT:}

- \(F\) – an element of the \texttt{residue\_ring()}
- \texttt{report\_coefficients} – whether to return the coefficients of the \texttt{phi()}-adic expansion or the actual polynomial (default: \texttt{False}, i.e., return the polynomial)

\textbf{OUTPUT:}

A polynomial in the domain of the valuation with reduction \(F\), monic if \(F\) is monic.

\textbf{ALGORITHM:}

Since this is the inverse of \texttt{reduce()}, we only have to go backwards through the algorithm described there.

\textbf{EXAMPLES:}

\begin{verbatim}
sage: # needs sage.libs.ntl/nsage: R.<u> = Qq(4, 10)
sage: S.<x> = R[]
sage: v = GaussValuation(S)
sage: w = v.augmentation(x^2 + x + u, 1/2)
sage: y = w.residue_ring().gen()
sage: u1 = w.residue_ring().base().gen()
sage: w.lift(1)
1 + O(2^10)
sage: w.lift(0)
0
sage: w.lift(u1)
(1 + O(2^10))*x
sage: w.reduce(w.lift(y)) == y
True
sage: w.reduce(w.lift(y + u1 + 1)) == y + u1 + 1
True
sage: ww = w.augmentation((x^2 + x + u)^2 + 2, 5/3)
sage: y = ww.residue_ring().gen()
sage: u2 = ww.residue_ring().base().gen()
sage: ww.reduce(ww.lift(y)) == y
True
sage: ww.reduce(ww.lift(1)) == 1
True
sage: ww.reduce(ww.lift(y + 1)) == y + 1
True
>>> from sage.all import *

>>> # needs sage.libs.ntl

>>> R = Qq(Integer(4), Integer(10), names=('u',)); (u,) = R._first_ngens(1)
>>> S = R['x']; (x,) = S._first_ngens(1)
>>> v = GaussValuation(S)

>>> w = v.augmentation(x*x*Integer(2) + x + u, Integer(1)/Integer(2))

>>> y = w.residue_ring().gen()

>>> u1 = w.residue_ring().base().gen()

>>> w.lift(Integer(1))
1 + O(2^10)

>>> w.lift(Integer(0))
0

>>> w.lift(u1)

(continues on next page)
(1 + O(2^10))*x
>>> w.reduce(w.lift(y)) == y
True
>>> w.reduce(w.lift(y + u1 + Integer(1))) == y + u1 + Integer(1)
True
>>> ww = w.augmentation((x**Integer(2) + x + u)**Integer(2),
˓→Integer(5)/Integer(3))
>>> y = ww.residue_ring().gen()
>>> u2 = ww.residue_ring().base().gen()
>>> ww.reduce(w.lift(y)) == y
True
>>> ww.reduce(ww.lift(y + u1 + Integer(1))) == y + u1 + Integer(1)
True
A more complicated example:

```python
sage: # needs sage.libs.ntl
sage: v = GaussValuation(S)
```

```python
sage: w = v.augmentation(x^2 + x + u, 1)
```

```python
sage: ww = w.augmentation((x^2 + x + u)^2 + 2*x*(x^2 + x + u) + 4*x, 3)
```

```python
sage: u = ww.residue_ring().base().gen()
```

```python
sage: F = ww.residue_ring()(u); F
u2
```

```python
sage: f = ww.lift(F); f
(2^-1 + O(2^9))*x^2 + (2^-1 + O(2^9))*x + u*2^-1 + O(2^9)
```

```python
sage: F == ww.reduce(f)
True
```

```python
>>> from sage.all import *
```

```python
>>> # needs sage.libs.ntl
```

```python
>>> v = GaussValuation(S)
```

```python
>>> w = v.augmentation(x^2 + x + u, Integer(1))
```

```python
>>> ww = w.augmentation((x^2 + x + u)^2 + (2^-1 + O(2^9))*x + u*2^-1 + O(2^9)
```

```python
sage: F == ww.reduce(f)
True
```

**lift_to_key** *(F, check=True)*

Lift the irreducible polynomial $F$ to a key polynomial.

**INPUT:**

- $F$ – an irreducible non-constant polynomial in the $\text{residue\_ring()}$ of this valuation
- `check` – whether or not to check correctness of $F$ (default: `True`)

**OUTPUT:**

A polynomial $f$ in the domain of this valuation which is a key polynomial for this valuation and which, for a suitable equivalence unit $R$, satisfies that the reduction of $Rf$ is $F$.

**ALGORITHM:**

5.8. Augmented valuations on polynomial rings 105
We follow the algorithm described in Theorem 13.1 [Mac1936I] which, after a \( \text{lift}() \) of \( \mathbb{F} \), essentially shifts the valuations of all terms in the \( \phi \)-adic expansion up and then kills the leading coefficient.

**EXAMPLES:**

```python
sage: # needs sage.libsntl
sage: R.<u> = Qq(4, 10)
sage: S.<x> = R[]
sage: v = GaussValuation(S)

sage: w = v.augmentation(x^2 + x + u, 1/2)

sage: y = w.residue_ring().gen()

sage: f = w.lift_to_key(y + 1); f
(1 + O(2^10))*x^4 + (2 + O(2^11))*x^3 + (1 + u*2 + O(2^10))*x^2 + (u*2 + O(2^11))*x + (u + 1) + u*2 + O(2^10)

sage: w.is_key(f)
True
```

A more complicated example:

```python
sage: # needs sage.libsntl
sage: v = GaussValuation(S)

sage: w = v.augmentation(x^2 + x + u, 1)

sage: ww = w.augmentation((x^2 + x + u)^2 + 2*x*(x^2 + x + u) + 4*x, 3)

sage: y = ww.residue_ring().gen()

sage: f = ww.lift_to_key(y^3+y+u)

sage: f.degree()
12

sage: ww.is_key(f)
True
```

```python
>>> from sage.all import *

>>> V = GaussValuation(S)

>>> w = V.augmentation(x^2 + x + u, 1)

>>> WW = w.augmentation((x^2 + x + u)^2 + 2*x*(x^2 + x + u) + 4*x, 3)

>>> y = WW.residue_ring().gen()

>>> f = WW.lift_to_key(y^3+y+u)

>>> f.degree()
12

>>> WW.is_key(f)
True
```
\textbf{reduce} \((f, \text{check}=\text{True}, \text{degree\_bound}=%\text{None}, \text{coefficients}=%\text{None}, \text{valuations}=%\text{None})\)

Reduce \(f\) module this valuation.

\textbf{INPUT}:

- \(f\) – an element in the domain of this valuation
- \(\text{check}\) – whether or not to check whether \(f\) has non-negative valuation (default: True)
- \(\text{degree\_bound}\) – an a-priori known bound on the degree of the result which can speed up the computation (default: not set)
- \(\text{coefficients}\) – the coefficients of \(f\) as produced by \text{coefficients()}\) or None (default: None); this can be used to speed up the computation when the expansion of \(f\) is already known from a previous computation.
- \(\text{valuations}\) – the valuations of \text{coefficients} or None (default: None)

\textbf{OUTPUT}:

an element of the \text{residue\_ring()}\) of this valuation, the reduction modulo the ideal of elements of positive valuation

\textbf{ALGORITHM}:

We follow the algorithm given in the proof of Theorem 12.1 of [Mac1936I]: If \(f\) has positive valuation, the reduction is simply zero. Otherwise, let \(f = \sum f_i \phi^i\) be the expansion of \(f\), as computed by \text{coefficients()}\). Since the valuation is zero, the exponents \(i\) must all be multiples of \(\tau\), the index the value group of the base valuation in the value group of this valuation. Hence, there is an \text{equivalence\_unit()}\) \(Q\) with the same valuation as \(\phi^\tau\). Let \(Q'\) be its \text{equivalence\_reciprocal()}\). Now, rewrite each term \(f_i \phi^i = (f_i Q')(\phi^\tau Q'^{-1})\); it turns out that the second factor in this expression is a lift of the generator of the \text{residue\_field()}\). The reduction of the first factor can be computed recursively.

\textbf{EXAMPLES}:

```python
sage: # needs sage.libs.ntl
sage: R.<u> = Qq(4, 10)
sage: S.<x> = R[]
sage: v = GaussValuation(S)
sage: v.reduce(x)
x
sage: v.reduce(S(u))
u0
sage: w = v.augmentation(x^2 + x + u, 1/2)
sage: w.reduce(S.one())
1
sage: w.reduce(S(2))
0
sage: w.reduce(S(u))
u0
sage: w.reduce(x)  # this gives the generator of the residue field extension...
\rightarrow \text{of } w \text{ over } v
u1
sage: f = (x^2 + x + u)^2 / 2
sage: w.reduce(f)
x
sage: w.reduce(f + x + 1)
x + u + 1
sage: g = ((x^2 + x + u)^2 + 2, 5/3)
sage: g
((x^2 + x + u)^2 + 2, 5/3)
sage: w.reduce(g)
```

(continues on next page)
Discrete Valuations and Discrete Pseudo-Valuations, Release 10.4

(resume from previous page)

>>> from sage.all import *
>>> # needs sage.libs.ntl
>>> R = Qq(Integer(4), Integer(10), names=('u',)); (u,) = R._first_ngens(1)
>>> S = R['x']; (x,) = S._first_ngens(1)
>>> v = GaussValuation(S)
>>> v.reduce(x)
x
>>> v.reduce(S(u))
u0
>>> w = v.augmentation(x**Integer(2) + x + u, Integer(1)/Integer(2))
>>> w.reduce(S.one())
1
>>> w.reduce(S(Integer(2)))
0
>>> w.reduce(S(u))
u0
>>> w.reduce(x)  # this gives the generator of the residue field extension of w over v
u1
>>> f = (x**Integer(2) + x + u)**Integer(2) / Integer(2)
>>> w.reduce(f)
x
>>> w.reduce(f + x + Integer(1))
x + u1 + 1
>>> ww = w.augmentation((x**Integer(2) + x + u)**Integer(2) + Integer(2),
                        Integer(5)/Integer(3))
>>> g = ((x**Integer(2) + x + u)**Integer(2) + Integer(2))**Integer(3) /
                        Integer(2)**Integer(3)
>>> ww.reduce(g)
x
>>> ww.reduce(f)
1
>>> ww.is_equivalent(f, Integer(1))
True
>>> ww.reduce(f * g)
x
>>> ww.reduce(f + g)
x + 1

residue_ring()

Return the residue ring of this valuation, i.e., the elements of non-negative valuation modulo the elements of positive valuation.

EXAMPLES:

sage: R.<x> = QQ[]

(continued on next page)
Discrete Valuations and Discrete Pseudo-Valuations, Release 10.4

sage: v = GaussValuation(R, QQ.valuation(2))
sage: w = v.augmentation(x^2 + x + 1, 1)
sage: w.residue_ring()  #...  
Univariate Polynomial Ring in x over Finite Field in u1 of size 2^2

Since trivial valuations of finite fields are not implemented, the resulting ring might be identical to the residue ring of the underlying valuation:

sage: w = v.augmentation(x, 1)
sage: w.residue_ring()  
Univariate Polynomial Ring in x over Finite Field of size 2 (using ...)

5.8. Augmented valuations on polynomial rings 109

class sage.rings.valuation.augmented_valuation.NonFinalFiniteAugmentedValuation(par-ent, v, phi, mu)  

Bases: FiniteAugmentedValuation, NonFinalAugmentedValuation

An augmented valuation which is discrete, i.e., which assigns a finite valuation to its last key polynomial, and which can be augmented further.

EXAMPLES:

sage: R.<x> = QQ[]
sage: v = GaussValuation(R, QQ.valuation(2))
sage: w = v.augmentation(x, 1)

>>> from sage.all import *  
>>> R = QQ['x']; (x,) = R._first_ngens(1)  
>>> v = GaussValuation(R, QQ.valuation(Integer(2)))  
>>> w = v.augmentation(x**Integer(2) + x + Integer(1), Integer(1))  
>>> w.residue_ring()  #...  
Univariate Polynomial Ring in x over Finite Field in u1 of size 2^2

Univariate Polynomial Ring in x over Finite Field of size 2 (using ...)

Univariate Polynomial Ring in x over Finite Field of size 2 (using ...)
5.9 Valuations which are defined as limits of valuations.

The discrete valuation of a complete field extends uniquely to a finite field extension. This is not the case anymore for fields which are not complete with respect to their discrete valuation. In this case, the extensions essentially correspond to the factors of the defining polynomial of the extension over the completion. However, these factors only exist over the completion and this makes it difficult to write down such valuations with a representation of finite length.

More specifically, let $v$ be a discrete valuation on $K$ and let $L = K[x]/(G)$ a finite extension thereof. An extension of $v$ to $L$ can be represented as a discrete pseudo-valuation $w'$ on $K[x]$ which sends $G$ to infinity. However, such $w'$ might not be described by an augmented valuation over a Gauss valuation anymore. Instead, we may need to write it as a limit of augmented valuations.

The classes in this module provide the means of writing down such limits and resulting valuations on quotients.

AUTHORS:
- Julian Rüth (2016-10-19): initial version

EXAMPLES:

In this function field, the unique place of $K$ which corresponds to the zero point has two extensions to $L$. The valuations corresponding to these extensions can only be approximated:

```python
sage: # needs sage.rings.function_field
sage: K.<x> = FunctionField(QQ)
sage: R.<y> = K[]

>>> from sage.all import *
```

```python
sage: L.<y> = K.extension(y^2 - x)
sage: v = K.valuation(1)
sage: w = v.extensions(L); w
[[ (x - 1)-adic valuation, v(y + 1) = 1 ]-adic valuation,
  (x - 1)-adic valuation, v(y - 1) = 1 ]-adic valuation]
```

The same phenomenon can be observed for valuations on number fields:

```python
sage: # needs sage.rings.number_field
```

```python
sage: K = QQ
sage: R.<t> = K[]

>>> from sage.all import *
```

```python
sage: L = K.extension(t^2 + 1)
sage: v = QQ.valuation(5)
sage: w = v.extensions(L); w
[[ 5-adic valuation, v(t + 2) = 1 ]-adic valuation,
  5-adic valuation, v(t + 3) = 1 ]-adic valuation]
```

(continues on next page)
Note: We often rely on approximations of valuations even if we could represent the valuation without using a limit. This is done to improve performance as many computations already can be done correctly with an approximation.

REFERENCES:
Limits of inductive valuations are discussed in [Mac1936I] and [Mac1936II]. An overview can also be found in Section 4.6 of [Rüt2014].
create_key (base_valuation, G)

Create a key from the parameters of this valuation.

EXAMPLES:

Note that this does not normalize base_valuation in any way. It is easily possible to create the same limit in two different ways:

```
sage: R.<x> = QQ[]
sage: v = GaussValuation(R, QQ.valuation(2))
sage: w = valuations.LimitValuation(v, x)  # indirect doctest
sage: v = v.augmentation(x, infinity)
sage: u = valuations.LimitValuation(v, x)
sage: u == w
False
```

The point here is that this is not meant to be invoked from user code. But mostly from other factories which have made sure that the parameters are normalized already.

create_object (version, key)

Create an object from key.

EXAMPLES:

```
sage: R.<x> = QQ[]
sage: v = GaussValuation(R, QQ.valuation(2))
sage: w = valuations.LimitValuation(v, x^2 + 1)  # indirect doctest
```

class sage.rings.valuation.limit_valuation.LimitValuation_generic (parent, approximation)
Bases: `DiscretePseudoValuation`

Base class for limit valuations.

A limit valuation is realized as an approximation of a valuation and means to improve that approximation when necessary.

**EXAMPLES:**

```sage
# needs sage.rings.function_field
K.<x> = FunctionField(QQ)
R.<y> = K[]
L.<y> = K.extension(y^2 - x)
v = K.valuation(0)
w = v.extension(L)
w._base_valuation
[ Gauss valuation induced by (x)-adic valuation, v(y) = 1/2 , ... ]
```

The currently used approximation can be found in the `_approximation` field:

```sage
w._base_valuation._approximation
[ Gauss valuation induced by (x)-adic valuation, v(y) = 1/2 ]
```

### reduce(f, check=True)

Return the reduction of `f` as an element of the `residue_ring()`.

**INPUT:**

- `f` – an element in the domain of this valuation of non-negative valuation
- `check` – whether or not to check that `f` has non-negative valuation (default: `True`)

**EXAMPLES:**

```sage
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)
v = K.valuation(Integer(0))
w = v.extension(L)
w._base_valuation
w._base_valuation._approximation
from sage.all import *
```

5.9. Valuations which are defined as limits of valuations.
Discrete Valuations and Discrete Pseudo-Valuations, Release 10.4

```python
>>> 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(1)), names=('y',)); (y,) = L._first_ngens(1)
>>> v = K.valuation(Integer(0))
>>> w = v.extension(L)
>>> w.reduce(y) # indirect doctest
u1
```

```python
class sage.rings.valuation.limit_valuation.MacLaneLimitValuation(parent, approximation, G)
Bases: LimitValuation_generic, InfiniteDiscretePseudoValuation
A limit valuation that is a pseudo-valuation on polynomial ring \( K[x] \) which sends a square-free polynomial \( G \) to infinity.

This uses the MacLane algorithm to compute the next element in the limit.

It starts from a first valuation \( \text{approximation} \) which has a unique augmentation that sends \( G \) to infinity and whose uniformizer must be a uniformizer of the limit and whose residue field must contain the residue field of the limit.

EXAMPLES:
```
sage: # needs sage.rings.number_field
sage: R.<x> = QQ[]
sage: K.<i> = QQ.extension(x^2 + 1)
sage: v = K.valuation(2)
sage: u = v._base_valuation; u
[ Gauss valuation induced by 2-adic valuation, v(x + 1) = 1/2 , ... ]
```
```
>>> from sage.all import *
>>> # needs sage.rings.number_field
>>> R = QQ['x']; (x,) = R._first_ngens(1)
>>> K = QQ.extension(x**Integer(2) + Integer(1), names=('i',)); (i,) = K._first_ngens(1)
>>> v = K.valuation(2)
>>> u = v._base_valuation; u
[ Gauss valuation induced by 2-adic valuation, v(x + 1) = 1/2 , ... ]
```

```
element_with_valuation(s)
Return an element with valuation \( s \).
```
```
extensions(ring)
Return the extensions of this valuation to \( \text{ring} \).
```
```
EXAMPLES:
```
sage: # needs sage.rings.number_field
sage: v = GaussianIntegers().valuation(2)
sage: u = v._base_valuation
sage: u.extensions(QQ['x'])
[[ Gauss valuation induced by 2-adic valuation, v(x + 1) = 1/2 , ... ]]```
is_negative_pseudo_valuation()

Return whether this valuation attains $-\infty$.

EXAMPLES:

For a Mac Lane limit valuation, this is never the case, so this method always returns False:

```python
sage: # needs sage.rings.number_field
sage: K = QQ
sage: R.<t> = K[]
sage: L.<t> = K.extension(t^2 + 1)
sage: v = QQ.valuation(2)
sage: u = v.extension(L)
>>> u.is_negative_pseudo_valuation()
False
```

lift($F$)

Return a lift of $F$ from the residue_ring() to the domain of this valuation.

EXAMPLES:

```python
sage: # needs sage.rings.function_field
sage: K.<x> = FunctionField(QQ)
sage: R.<y> = K[]
sage: L.<y> = K.extension(y^4 - x^2 - 2*x - 1)
>>> w = v.extensions(L)[1]; w
( x - 1)-adic valuation, v(y^2 - 2) = 1 ]-adic valuation
sage: s = w.reduce(y); s
u1
sage: w.lift(s)  # indirect doctest
y
```
>>> v = K.valuation(Integer(1))
>>> w = v.extensions(L)[Integer(1)]; w
[(x - 1)-adic valuation, v(y^2 - 2) = 1 ]-adic valuation
>>> s = w.reduce(y); s
u1
>>> w.lift(s) # indirect doctest
y

`lower_bound(f)`

Return a lower bound of this valuation at x.

Use this method to get an approximation of the valuation of x when speed is more important than accuracy.

**EXAMPLES:**

```python
sage: # needs sage.rings.number_field
sage: K = QQ
sage: R.<t> = K[]
sage: L.<t> = K.extension(t^2 + 1)
sage: v = QQ.valuation(2)
sage: u = v.extension(L)
sage: u.lower_bound(Integer(1024)*t + Integer(1024))
10
sage: u(Integer(1024)*t + Integer(1024))
21/2
```

```python
>>> from sage.all import *
>>> # needs sage.rings.number_field
>>> K = QQ
>>> R = K[t]; (t,) = R._first_ngens(1)
>>> L = K.extension(t^Integer(2) + Integer(1), names =(t,)); (t,) = L._
˓→first_ngens(1)
>>> v = QQ.valuation(Integer(2))
>>> u = v.extension(L)
>>> u.lower_bound(Integer(1024)*t + Integer(1024))
10
>>> u(Integer(1024)*t + Integer(1024))
21/2
```

`residue_ring()`

Return the residue ring of this valuation, which is always a field.

**EXAMPLES:**

```python
sage: # needs sage.rings.number_field
sage: K = QQ
sage: R.<t> = K[]
sage: L.<t> = K.extension(t^2 + 1)
sage: v = QQ.valuation(2)
sage: w = v.extension(L)
sage: w.residue_ring()
Finite Field of size 2
```

```python
>>> from sage.all import *
>>> # needs sage.rings.number_field
>>> K = QQ
```
Discrete Valuations and Discrete Pseudo-Valuations, Release 10.4

(continued from previous page)

```python
>>> R = K['t']; (t,) = R._first_ngens(1)
>>> L = K.extension(t**Integer(2) + Integer(1), names=('t',)); (t,) = L._
˓→first_ngens(1)
>>> v = QQ.valuation(Integer(2))
>>> w = v.extension(L)
>>> w.residue_ring()
Finite Field of size 2
```

**restriction** *(ring)*

Return the restriction of this valuation to *ring*.

**EXAMPLES:**

```
sage: # needs sage.rings.number_field
sage: K = QQ
sage: R.<t> = K[]
sage: L.<t> = K.extension(t^2 + 1)
sage: v = QQ.valuation(2)
sage: w = v.extension(L)
sage: w._base_valuation.restriction(K)
2-adic valuation
```

**simplify** *(f, error=None, force=False)*

Return a simplified version of *f*.

Produce an element which differs from *f* by an element of valuation strictly greater than the valuation of *f* (or strictly greater than *error* if set.)

**EXAMPLES:**

```
sage: # needs sage.rings.number_field
sage: K = QQ
sage: R = K['t']; (t,) = R._first_ngens(1)
sage: L = K.extension(t**Integer(2) + Integer(1), names=('t',)); (t,) = L._
˓→first_ngens(1)
sage: v = QQ.valuation(Integer(2))
sage: u = v.extension(L)
sage: u.simplify(t + 1024, force=True)
t
```

(continues on next page)
Discrete Valuations and Discrete Pseudo-Valuations, Release 10.4

>>> v = QQ.valuation(Integer(2))
>>> u = v.extension(L)
>>> u.simplify(t + Integer(1024), force=True)
t

`uniformizer()`

Return a uniformizing element for this valuation.

**EXAMPLES:**

```python
sage: 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, names=('y',)); (y,) = L._first_ngens(1)
v = K.valuation(Integer(0))
w = v.extension(L)
w.uniformizer()  # indirect doctest
y
```

`upper_bound(f)`

Return an upper bound of this valuation at `x`.

Use this method to get an approximation of the valuation of `x` when speed is more important than accuracy.

**EXAMPLES:**

```python
sage: from sage.all import *
# needs sage.rings.number_field
K = QQ
R = K[t]; (t,) = R._first_ngens(1)
L = K.extension(t**Integer(2) + Integer(1), names=('t',)); (t,) = L._first_ngens(1)
v = QQ.valuation(Integer(2))
u = v.extension(L)
u.upper_bound(Integer(1024)*t + Integer(1024))
```

(continues on previous page)

(continues on next page)
value_semigroup()

Return the value semigroup of this valuation.

5.10 Valuations which are implemented through a map to another valuation

EXAMPLES:

Extensions of valuations over finite field extensions \( L = K[x]/(G) \) are realized through an infinite valuation on \( K[x] \) which maps \( G \) to infinity:

```python
sage: # needs sage.rings.function_field
sage: K.<x> = FunctionField(QQ)
```

```python
sage: R.<y> = K[
```

```python
sage: L.<y> = K.extension(y^2 - x)
```

```python
sage: v = K.valuation(0)
```

```python
sage: w = v.extension(L); w
```

```python
(x)-adic valuation
```

```python
sage: w._base_valuation
```

```python
[ Gauss valuation induced by (x)-adic valuation, v(y) = 1/2 , ... ]
```

AUTHORS:

- Julian Rüth (2016-11-10): initial version
class sage.rings.valuation.mapped_valuation.FiniteExtensionFromInfiniteValuation (parent, base_valuation)

Bases: MappedValuation_base, DiscreteValuation

A valuation on a quotient of the form \( L = K[x]/(G) \) with an irreducible \( G \) which is internally backed by a pseudo-valuations on \( K[x] \) which sends \( G \) to infinity.

INPUT:

- parent – the containing valuation space (usually the space of discrete valuations on \( L \))
- base_valuation – an infinite valuation on \( K[x] \) which takes \( G \) to infinity

EXAMPLES:

```python
sage: # needs sage.rings.function_field
sage: K.<x> = FunctionField(QQ)
sage: R.<y> = K[]
sage: L.<y> = K.extension(y^2 - x)
sage: v = K.valuation(0)
sage: w = v.extension(L); w
(x)-adic valuation
```

```
>>> from sage.all import *
>>> # needs sage.rings.function_field
>>> K = FunctionField(QQ, names=(x,)); (x,) = K._first_ngens(1)
>>> R = K[t]; (t,) = R._first_ngens(1)
>>> L = K.extension(t^2 + 1, names=(y,)); (y,) = L._first_ngens(1)
>>> v = K.valuation(t)
>>> w = v.extension(L); w
(x)-adic valuation
```

**lower_bound** \((x)\)

Return a lower bound of this valuation at \( x \).

Use this method to get an approximation of the valuation of \( x \) when speed is more important than accuracy.

EXAMPLES:

```python
sage: # needs sage.rings.number_field
sage: K = QQ
sage: R.<t> = K[]
```

```python
>>> from sage.all import *
```

```
>>> K = QQ
```

```
>>> # needs sage.rings.function_field
```

```
>>> L.<t> = K.extension(t^2 + 1)
```

```
>>> v = valuations.pAdicValuation(QQ, 5)
>>> u, uu = v.extensions(L)
```

```python
sage: u.lower_bound(t + 2)
0
```

```python
sage: u(t + 2)
1
```

```python
>>> from sage.all import *
```

```python
>>> # needs sage.rings.number_field
>>> K = QQ
```
Discrete Valuations and Discrete Pseudo-Valuations, Release 10.4

(continued from previous page)

```python
>>> L = K.extension(t**Integer(2) + Integer(1), names=('t',)); (t,) = L._
˓→first_ngens(1)
>>> v = valuations.pAdicValuation(QQ, Integer(5))
>>> u,uu = v.extensions(L)
>>> u.lower_bound(t + Integer(2))
0
>>> u(t + Integer(2))
1
restriction(ring)

Return the restriction of this valuation to ring.

EXAMPLES:

```python
sage: # needs sage.rings.number_field
sage: K = QQ
sage: R.<t> = K[]
sage: L.<t> = K.extension(t^2 + 1)
```

```python
>>> v = valuations.pAdicValuation(QQ, 2)
```

```python
>>> w = v.extension(L)
```

```python
>>> w.restriction(K) is v
True
```

simplify(x, error=None, force=False)

Return a simplified version of x.

Produce an element which differs from x by an element of valuation strictly greater than the valuation of x (or strictly greater than error if set.)

EXAMPLES:

```python
sage: # needs sage.rings.number_field
sage: K = QQ
sage: R.<t> = K[]
```

```python
>>> L = K.extension(t**Integer(2) + Integer(1), names=('t',)); (t,) = L._
˓→first_ngens(1)
```

```python
>>> v = valuations.pAdicValuation(QQ, 5)
```

```python
>>> u,uu = v.extensions(L)
```

```python
>>> f = 125*t + 1
```

```python
>>> u.simplify(f, error=u(f), force=True)
```

```python
1
```

```python
>>> from sage.all import *
```

```python
# needs sage.rings.number_field
```

```python
K = QQ
```

```python
R = K[t]; (t,) = R._first_ngens(1)
```

```python
L = K.extension(t**Integer(2) + Integer(1), names=('t',)); (t,) = L._
˓→first_ngens(1)
```

```python
>>> v = valuations.pAdicValuation(QQ, Integer(5))
```

```python
>>> u,uu = v.extensions(L)
```

```python
>>> u(t + Integer(2))
```

```python
1
```

(continues on next page)
Discrete Valuations and Discrete Pseudo-Valuations, Release 10.4

(continued from previous page)

```python
>>> L = K.extension(t**Integer(2) + Integer(1), names=('t',)); (t,) = L._first_ngens(1)
>>> v = valuations.pAdicValuation(QQ, Integer(5))
>>> u,uu = v.extensions(L)
>>> f = Integer(125)*t + Integer(1)
>>> u.simplify(f, error=u(f), force=True)
1
```

**upper_bound** \((x)\)

Return an upper bound of this valuation at \(x\).

Use this method to get an approximation of the valuation of \(x\) when speed is more important than accuracy.

**EXAMPLES:**

```python
sage: # needs sage.rings.number_field
sage: K = QQ
sage: R.<t> = K[]
sage: L.<t> = K.extension(t^2 + 1)
sage: v = valuations.pAdicValuation(QQ, 5)
sage: u,uu = v.extensions(L)
sage: u.upper_bound(t + Integer(2)) >= 1
True
sage: u(t + 2)
1
```

```python
>>> from sage.all import *
```

```python
>>> # needs sage.rings.number_field
>>> K = QQ
>>> R = K[t]; (t,) = R._first_ngens(1)
>>> L = K.extension(t**Integer(2) + Integer(1), names=('t',)); (t,) = L._first_ngens(1)
>>> v = valuations.pAdicValuation(QQ, Integer(5))
>>> u,uu = v.extensions(L)
>>> u.upper_bound(t + Integer(2)) >= Integer(1)
True
>>> u(t + Integer(2))
1
```

class `sage.rings.valuation.mapped_valuation.FiniteExtensionFromLimitValuation`

**Bases:** `FiniteExtensionFromInfiniteValuation`  

An extension of a valuation on a finite field extensions \(L = K[x]/(G)\) which is induced by an infinite limit valuation on \(K[x]\).

**EXAMPLES:**
Discrete Valuations and Discrete Pseudo-Valuations, Release 10.4

```sage
# needs sage.rings.function_field
sage: K.<x> = FunctionField(QQ)
```

```sage
sage: R.<y> = K[]
```

```sage
sage: L.<y> = K.extension(y^2 - x)
```

```sage
sage: v = K.valuation(1)
```

```sage
sage: w = v.extensions(L); w
[[ (x - 1)-adic valuation, v(y + 1) = 1 ]-adic valuation,
 [ (x - 1)-adic valuation, v(y - 1) = 1 ]-adic valuation]
```

```sage
>>> from sage.all import *
```

```sage
>>> # needs sage.rings.function_field
```

```sage
>>> K = FunctionField(QQ, names=('x',)); (x,) = K._first_ngens(1)
```

```sage
>>> R = K['y']; (y,) = R._first_ngens(1)
```

```sage
>>> L = K.extension(y**Integer(2) - x, names=('y',)); (y,) = L._first_ngens(1)
```

```sage
>>> v = K.valuation(Integer(1))
```

```sage
>>> w = v.extensions(L); w
[[ (x - 1)-adic valuation, v(y + 1) = 1 ]-adic valuation,
 [ (x - 1)-adic valuation, v(y - 1) = 1 ]-adic valuation]
```

```sage
class sage.rings.valuation.mapped_valuation.MappedValuation_base(parent, base_valuation)
```

Bases: `DiscretePseudoValuation`

A valuation which is implemented through another proxy “base” valuation.

EXAMPLES:

```sage
sage: # needs sage.rings.function_field
sage: K.<x> = FunctionField(QQ)
```

```sage
sage: R.<y> = K[]
```

```sage
sage: L.<y> = K.extension(y^2 - x)
```

```sage
sage: v = K.valuation(0)
```

```sage
sage: w = v.extensions(L); w
(x)-adic valuation
```

```sage
sage: v.element_with_valuation(1)
```

```sage
5
```

5.10. Valuations which are implemented through a map to another valuation

123
Discrete Valuations and Discrete Pseudo-Valuations, Release 10.4

```python
>>> from sage.all import *
>>> # needs sage.rings.number_field
>>> K = QQ
>>> R = K['t']; (t,) = R._first_ngens(1)
>>> L = K.extension(t**Integer(2) + Integer(1), names=('t',)); (t,) = L._first_ngens(1)
>>> v = valuations.pAdicValuation(QQ, Integer(5))
>>> u, uu = v.extensions(L)
>>> u.element_withvaluation(Integer(1))
5
```

**lift** (*F*)

Lift *F* from the `residue_field()` of this valuation into its domain.

**EXAMPLES:**

```python
sage: # needs sage.rings.function_field
sage: K.<x> = FunctionField(QQ)
sage: R.<y> = K[]
sage: L.<y> = K.extension(y^2 - x)
sage: v = K.valuation(2)
sage: w = v.extension(L)
sage: w.lift(w.residue_field().gen())
y
```

**reduce** (*f*)

Return the reduction of *f* in the `residue_field()` of this valuation.

**EXAMPLES:**

```python
sage: # needs sage.rings.function_field
sage: K.<x> = FunctionField(QQ)
sage: R.<y> = K[]
sage: L.<y> = K.extension(y^2 - (x - 2))
sage: v = K.valuation(0)
sage: w = v.extension(L)
sage: w.reduce(y)
y
```
w.reduce(y)

residue_ring()

Return the residue ring of this valuation.

EXAMPLES:

```python
sage: K = QQ
sage: R.<t> = K[

# needs sage.rings.number_field
sage: L.<t> = K.extension(t^2 + 1)

# needs sage.rings.number_field
sage: v = valuations.pAdicValuation(QQ, 2)

# needs sage.rings.number_field
sage: v.extension(L).residue_ring()

Finite Field of size 2
```

simplify(x, error=None, force=False)

Return a simplified version of x.

Produce an element which differs from x by an element of valuation strictly greater than the valuation of x (or strictly greater than error if set.)

If force is not set, then expensive simplifications may be avoided.

EXAMPLES:

```python
sage: # needs sage.rings.function_field
sage: K.<x> = FunctionField(QQ)

# needs sage.rings.function_field
sage: R.<y> = K[

# needs sage.rings.function_field
sage: L.<y> = K.extension(y^2 - x)

# needs sage.rings.function_field
sage: v = K.valuation(0)

# needs sage.rings.function_field
sage: w = v.extensions(L)[0]

>>> from sage.all import *

# needs sage.rings.function_field
>>> K = FunctionField(QQ, names=('x',))

# needs sage.rings.function_field
>>> (x,) = K._first_ngens(1)

# needs sage.rings.function_field
>>> R = K['y']; (y,) = R._first_ngens(1)

# needs sage.rings.function_field
>>> L = K.extension(y**Integer(2) - x, names=('y',)); (y,) = L._first_ngens(1)

# needs sage.rings.function_field
>>> v = K.valuation(Integer(0))

# needs sage.rings.function_field
>>> w = v.extensions(L)[Integer(0)]

As _relative_size() misses the bloated term x^32, the following term does not get simplified:

```python
sage: w.simplify(y + x^32)

# needs sage.rings.function_field
y + x^32
```

5.10. Valuations which are implemented through a map to another valuation
In this case the simplification can be forced but this should not happen as a default as the recursive simplification can be quite costly:

```python
sage: w.simplify(y + x^32, force=True)  # needs sage.rings.function_field
y
```

**uniformizer()**

Return a uniformizing element of this valuation.

**EXAMPLES:**

```python
sage: K = QQ
sage: R.<t> = K[]
sage: L.<t> = K.extension(t^2 + 1)  # needs sage.rings.number_field
sage: v = valuations.pAdicValuation(QQ, 2)
sage: v.extension(L).uniformizer()  # needs sage.rings.number_field
t + 1
```

5.11 Valuations which are scaled versions of another valuation

**EXAMPLES:**

```python
sage: 3 * ZZ.valuation(3)
3 * 3-adic valuation
```

**AUTHORS:**

- Julian Rüth (2016-11-10): initial version
class sage.rings.valuation.scaled_valuation.ScaledValuationFactory

Bases: UniqueFactory

Return a valuation which scales the valuation base by the factor s.

EXAMPLES:

```python
sage: 3*ZZ.valuation(2)  # indirect doctest
3 * 2-adic valuation

>>> from sage.all import *
>>> Integer(3)*ZZ.valuation(Integer(2))  # indirect doctest
3 * 2-adic valuation
```

create_key (base, s)

Create a key which uniquely identifies a valuation.

create_object (version, key)

Create a valuation from key.

class sage.rings.valuation.scaled_valuation.ScaledValuation_generic (parent, base_valuation, s)

Bases: DiscreteValuation

A valuation which scales another base_valuation by a finite positive factor s.

EXAMPLES:

```python
sage: v = 3*ZZ.valuation(3); v
3 * 3-adic valuation

>>> from sage.all import *
>>> v = Integer(3)*ZZ.valuation(Integer(3)); v
3 * 3-adic valuation

extensions (ring)

Return the extensions of this valuation to ring.

EXAMPLES:

```python
sage: v = 3*ZZ.valuation(5)
sage: v.extensions(GaussianIntegers().fraction_field())  # indirect doctest
[3 * 5-adic valuation, v(x + 2) = 1 ]-adic valuation,
3 * 5-adic valuation, v(x + 3) = 1 ]-adic valuation]

>>> from sage.all import *
>>> v = Integer(3)*ZZ.valuation(Integer(5)); v
3 * 3-adic valuation
>>> v.extensions(GaussianIntegers().fraction_field())  # indirect doctest
[3 * 5-adic valuation, v(x + 2) = 1 ]-adic valuation,
3 * 5-adic valuation, v(x + 3) = 1 ]-adic valuation]
```

lift (F)

Lift F from the residue_field() of this valuation into its domain.

EXAMPLES:
**reduce** (*f*)

Return the reduction of *f* in the *residue_field()* of this valuation.

**EXAMPLES:**

```
sage: v = 3*ZZ.valuation(2)
sage: v.reduce(1)
1
```

**residue_ring**()

Return the residue field of this valuation.

**EXAMPLES:**

```
sage: v = 3*ZZ.valuation(2)
sage: v.residue_ring()
Finite Field of size 2
```

**restriction** (*ring*)

Return the restriction of this valuation to *ring*.

**EXAMPLES:**

```
sage: v = 3*QQ.valuation(5)
sage: v.restriction(ZZ)
3 * 5-adic valuation
```

**uniformizer**()

Return a uniformizing element of this valuation.

**EXAMPLES:**

```
```
**5.12 Discrete valuations on function fields**

**AUTHORS:**
- Julian Rüth (2016-10-16): initial version

**EXAMPLES:**

We can create classical valuations that correspond to finite and infinite places on a rational function field:

```python
sage: K.<x> = FunctionField(QQ)
sage: v = K.valuation(1); v
(x - 1)-adic valuation
sage: v = K.valuation(x^2 + 1); v
(x^2 + 1)-adic valuation
sage: v = K.valuation(1/x); v
Valuation at the infinite place
```

Note that we can also specify valuations which do not correspond to a place of the function field:

```python
sage: R.<x> = QQ[]
sage: w = valuations.GaussValuation(R, QQ.valuation(2))
```

(continues on next page)
Valuations on a rational function field can then be extended to finite extensions:

```python
sage: v = K.valuation(x - 1); v
(x - 1)-adic valuation
sage: R.<y> = K[]
sage: L.<y> = K.extension(y^2 - x)  # needs sage.rings.function_field
sage: w = v.extensions(L); w
[[ (x - 1)-adic valuation, v(y + 1) = 1 ]-adic valuation,
  [ (x - 1)-adic valuation, v(y - 1) = 1 ]-adic valuation]
```

REFERENCES:

An overview of some computational tools relating to valuations on function fields can be found in Section 4.6 of [Rüt2014]. Most of this was originally developed for number fields in [Mac1936I] and [Mac1936II].

**class** `sage.rings.function_field.valuation.ClassicalFunctionFieldValuation_base` (parent)

Bases: `DiscreteFunctionFieldValuation_base`

Base class for discrete valuations on rational function fields that come from points on the projective line.

**class** `sage.rings.function_field.valuation.DiscreteFunctionFieldValuation_base` (parent)

Bases: `DiscreteValuation`

Base class for discrete valuations on function fields.

**extensions** (`L`)

Return the extensions of this valuation to `L`.

**EXAMPLES:**

```python
sage: K.<x> = FunctionField(QQ)
sage: v = K.valuation(x)
sage: R.<y> = K[]
```
Discrete Valuations and Discrete Pseudo-Valuations, Release 10.4

sage: L.<y> = K.extension(y^2 - x)  # needs sage.rings.function_field
sage: v.extensions(L)  # needs sage.rings.function_field
[(x)-adic valuation]

>>> 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)
"# needs sage.rings.function_field
>>> v.extensions(L)  # needs sage.rings.function_field
[(x)-adic valuation]

class sage.rings.function_field.valuation.FiniteRationalFunctionFieldValuation (parent, base_valuation)

Bases: InducedRationalFunctionFieldValuation_base, ClassicalFunctionFieldValuation_base, RationalFunctionFieldValuation_base

Valuation of a finite place of a function field.

EXAMPLES:

sage: K.<x> = FunctionField(QQ)
sage: v = K.valuation(x + 1); v  # indirect doctest
(x + 1)-adic valuation

>>> from sage.all import *
>>> K = FunctionField(QQ, names=('x',)); (x,) = K._first_ngens(1)
>>> v = K.valuation(x + Integer(1)); v  # indirect doctest
(x + 1)-adic valuation

A finite place with residual degree:

sage: w = K.valuation(x^2 + 1); w
(x^2 + 1)-adic valuation

>>> from sage.all import *
>>> w = K.valuation(x**Integer(2) + Integer(1)); w
(x^2 + 1)-adic valuation

A finite place with ramification:

sage: K.<t> = FunctionField(GF(3))
sage: L.<x> = FunctionField(K)
sage: u = L.valuation(x^3 - t); u
(x^3 + 2*t)-adic valuation

5.12. Discrete valuations on function fields
Discrete Valuations and Discrete Pseudo-Valuations, Release 10.4

```python
>>> from sage.all import *
>>> K = FunctionField(GF(Integer(3)), names=('t',)); (t,) = K._first_ngens(1)
>>> L = FunctionField(K, names=('x',)); (x,) = L._first_ngens(1)
>>> u = L.valuation(x**Integer(3) - t); u
(x^3 + 2*t)-adic valuation
```

A finite place with residual degree and ramification:

```python
sage: q = L.valuation(x^6 - t); q
(x^6 + 2*t)-adic valuation
```

```python
>>> from sage.all import *
>>> q = L.valuation(x**Integer(6) - t); q
(x^6 + 2*t)-adic valuation
```

class sage.rings.function_field.valuation.FunctionFieldExtensionMappedValuation(parent, base_valuation, to_base_valuation_domain, from_base_valuation_domain)

Bases: FunctionFieldMappedValuationRelative_base

A valuation on a finite extensions of function fields $L = K[y]/(G)$ where $K$ is another function field which redirects to another base_valuation on an isomorphism function field $M = K[y]/(H)$.

The isomorphisms must be trivial on $K$.

EXAMPLES:

```python
sage: K.<x> = FunctionField(GF(2))
sage: R.<y> = K[]  #...
  → needs sage.rings.function_field
sage: v = K.valuation(1/x)
sage: w = v.extension(L)  #...
  → needs sage.rings.function_field

sage: w(x)  #...
  → needs sage.rings.function_field
-1

sage: w(y)  #...
  → needs sage.rings.function_field
-3/2

sage: w.uniformizer()  #...
  → needs sage.rings.function_field
1/x^2*y
```
Discrete Valuations and Discrete Pseudo-Valuations, Release 10.4

```python
>>> 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) + y + x**Integer(3), names=('y',)); (y,) = L._first_ngens(1)  # needs sage.rings.function_field
>>> v = K.valuation(Integer(1)/x)
>>> w = v.extension(L)  # needs sage.rings.function_field
>>> w(x)  # needs sage.rings.function_field
-1
>>> w(y)  # needs sage.rings.function_field
-3/2
>>> w.uniformizer()  # needs sage.rings.function_field
1/x^2*y
```

**restriction**(ring)

Return the restriction of this valuation to ring.

**EXAMPLES:**

```python
sage: # needs sage.rings.function_field
sage: K.<x> = FunctionField(GF(2))
sage: R.<y> = K[]
sage: L.<y> = K.extension(y^2 + y + x^3)
sage: v = K.valuation(1/x)
sage: w = v.extension(L)
sage: w.restriction(K) is v
True
```

```python
>>> from sage.all import *
>>> # needs sage.rings.function_field
>>> 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) + y + x**Integer(3), names=('y',)); (y,) = L._first_ngens(1)
>>> v = K.valuation(Integer(1)/x)
>>> w = v.extension(L)
>>> w.restriction(K) is v
True
```

**class** sage.rings.function_field.valuation.FunctionFieldFromLimitValuation(parent, approximants, G, approximants)

A valuation on a finite extensions of function fields $L = K[y]/(G)$ where $K$ is another function field.

**EXAMPLES:**

5.12. Discrete valuations on function fields 133
scale (scalar)

Return this valuation scaled by scalar.

EXAMPLES:

```python
>>> 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 + Integer(1)), names=('y',))
>>> v = K.valuation(x - Integer(1)) # indirect doctest
>>> w = v.extension(L)
>>> 3*w
3 * (x - 1)-adic valuation
```

```python
>>> 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 + Integer(1)), names=('y',))
>>> v = K.valuation(x - Integer(1)) # indirect doctest
>>> w = v.extension(L)
>>> 3*w
3 * (x - 1)-adic valuation
```
class sage.rings.function_field.valuation.FunctionFieldMappedValuationRelative_base(parent, base_valuation, to_basevaluation, from_basevaluation_domain, from_basevaluation_domain)

Bases: FunctionFieldMappedValuation_base

A valuation on a function field which relies on a basevaluation on an isomorphic function field and which is such that the map from and to the other function field is the identity on the constant field.

EXAMPLES:

sage: K.<x> = FunctionField(GF(2))
sage: v = K.valuation(1/x); v
Valuation at the infinite place

>>> from sage.all import *
>>> K = FunctionField(GF(Integer(2)), names=('x',)); (x,) = K._first_ngens(1)
>>> v = K.valuation(Integer(1)/x); v
Valuation at the infinite place

restriction (ring)

Return the restriction of this valuation to ring.

EXAMPLES:

sage: K.<x> = FunctionField(GF(2))
sage: K.valuation(1/x).restriction(GF(2))
Trivial valuation on Finite Field of size 2

>>> from sage.all import *
>>> K = FunctionField(GF(Integer(2)), names=('x',)); (x,) = K._first_ngens(1)
>>> K.valuation(Integer(1)/x).restriction(GF(Integer(2)))
Trivial valuation on Finite Field of size 2
class sage.rings.function_field.valuation.FunctionFieldMappedValuation_base(parent, base_valuation, to_base_valuation_domain, from_base_valuation_domain)

Bases: FunctionFieldValuation_base, MappedValuation_base

A valuation on a function field which relies on a base_valuation on an isomorphic function field.

EXAMPLES:

```python
sage: K.<x> = FunctionField(GF(2))
sage: v = K.valuation(1/x); v
Valuation at the infinite place
```

```python
>>> from sage.all import *
>>> K = FunctionField(GF(Integer(2)), names=('x',)); (x,) = K._first_ngens(1)
>>> v = K.valuation(Integer(1)/x); v
Valuation at the infinite place
```

is_discrete_valuation()

Return whether this is a discrete valuation.

EXAMPLES:

```python
sage: # needs sage.rings.function_field
sage: K.<x> = FunctionField(QQ)
sage: R.<y> = K[]
sage: L.<y> = K.extension(y^2 - x^4 - 1)
sage: v = K.valuation(1/x)
sage: w0,w1 = v.extensions(L)
sage: w0.is_discrete_valuation()
True
```

```python
>>> 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(4) - Integer(1), names=('y',));
   (y,) = L._first_ngens(1)
>>> v = K.valuation(Integer(1)/x)
>>> w0,w1 = v.extensions(L)
>>> w0.is_discrete_valuation()
True
```

scale(scalar)

Return this valuation scaled by scalar.

EXAMPLES:
class sage.rings.function_field.valuation.FunctionFieldValuationFactory

Create a valuation on domain corresponding to prime.

INPUT:

- domain -- a function field
- 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:

```python
sage: K.<x> = FunctionField(QQ)
sage: v = K.valuation(1); v  # indirect doctest
(x - 1)-adic valuation
sage: v(x)
0
sage: v(x - 1)
1
```

```python
>>> from sage.all import *

>>> K = FunctionField(QQ, names=('x',)); (x,) = K._first_ngens(1)

>>> v = K.valuation(Integer(1)); v  # indirect doctest
(x - 1)-adic valuation

>>> v(x)
0

>>> v(x - Integer(1))
1
```

See sage.rings.function_field.function_field.FunctionField.valuation() for further examples.

5.12. Discrete valuations on function fields 137
create_key_and_extra_args \( (domain, \text{prime}) \)
Create a unique key which identifies the valuation given by \text{prime} on \text{domain}.

create_key_and_extra_args_from_place \( (domain, \text{generator}) \)
Create a unique key which identifies the valuation at the place specified by \text{generator}.

create_key_and_extra_args_fromvaluation \( (domain, \text{valuation}) \)
Create a unique key which identifies the valuation which extends \text{valuation}.

create_key_and_extra_args_fromvaluation_on_isomorphic_field \( (domain, \text{valuation}, \text{to_valuation_domain}, \text{from_valuation_domain}) \)
Create a unique key which identifies the valuation which is \text{valuation} after mapping through \text{to_valuation_domain}.

create_object \( (version, \text{key}, **\text{extra_args}) \)
Create the valuation specified by \text{key}.

EXAMPLES:

```python
sage: K.<x> = FunctionField(QQ)
sage: R.<x> = QQ[]
sage: w = valuations.GaussValuation(R, QQ.valuation(2))
sage: v = K.valuation(w); v  # indirect doctest
2-adic valuation
```

```python
>>> from sage.all import *
>>> K = FunctionField(QQ, names=(x,)); (x,) = K._first_ngens(1)
>>> R = QQ['x']; (x,) = R._first_ngens(1)
>>> w = valuations.GaussValuation(R, QQ.valuation(Integer(2)))
>>> v = K.valuation(w); v  # indirect doctest
2-adic valuation
```

class sage.rings.function_field.valuation.FunctionFieldValuation_base \( (\text{parent}) \)
Bases: DiscretePseudoValuation
Abstract base class for any discrete (pseudo-)valuation on a function field.

class sage.rings.function_field.valuation.InducedRationalFunctionFieldValuation_base \( (\text{parent}, \text{base_valu-}
    \text{ation}) \)
Bases: FunctionFieldValuation_base
Base class for function field valuation induced by a valuation on the underlying polynomial ring.

extensions \( (L) \)
Return all extensions of this valuation to \( L \) which has a larger constant field than the domain of this valuation.

EXAMPLES:
Discrete valuations and discrete pseudo-valuations

```python
sage: v = K.valuation(x^2 + 1)
sage: L.<x> = FunctionField(GaussianIntegers().fraction_field())
sage: v.extensions(L)  # indirect doctest
[(x - I)-adic valuation, (x + I)-adic valuation]
```

```python
>>> from sage.all import *
>>> # needs sage.rings.number_field
>>> K = FunctionField(QQ, names=('x',)); (x,) = K._first_ngens(1)
>>> v = K.valuation(x**Integer(2) + Integer(1))
>>> L = FunctionField(GaussianIntegers().fraction_field(), names=('x',)); (x, →) = L._first_ngens(1)
>>> v.extensions(L)  # indirect doctest
[(x - I)-adic valuation, (x + I)-adic valuation]
```

**lift** \( F \)

Return a lift of \( F \) to the domain of this valuation such that `reduce()` returns the original element.

**EXAMPLES:**

```python
sage: K.<x> = FunctionField(QQ)
sage: v = K.valuation(x)
sage: v.lift(0)
0
sage: v.lift(1)
1
```

```python
>>> from sage.all import *
>>> K = FunctionField(QQ, names=('x',)); (x,) = K._first_ngens(1)
>>> v = K.valuation(x**Integer(2) + Integer(1))
>>> v.lift(Integer(0))
0
>>> v.lift(Integer(1))
1
```

**reduce** \( f \)

Return the reduction of \( f \) in `residue_ring()`.

**EXAMPLES:**

```python
sage: K.<x> = FunctionField(QQ)
sage: v = K.valuation(x^2 + 1)
sage: v.reduce(x)  # needs sage.rings.number_field
u1
```

```python
>>> from sage.all import *
>>> K = FunctionField(QQ, names=('x',)); (x,) = K._first_ngens(1)
>>> v = K.valuation(x**Integer(2) + Integer(1))
>>> v.reduce(x)  # needs sage.rings.number_field
u1
```

**residue_ring**

Return the residue field of this valuation.

**EXAMPLES:**

```python
```
Discrete Valuations and Discrete Pseudo-Valuations, Release 10.4

```python
sage: K.<x> = FunctionField(QQ)
sage: K.valuation(x).residue_ring()
Rational Field

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

restriction(ring)
Return the restriction of this valuation to ring.

EXAMPLES:

```python
sage: K.<x> = FunctionField(QQ)
sage: K.valuation(x).restriction(QQ)
Trivial valuation on Rational Field

>>> from sage.all import *
>>> K = FunctionField(QQ, names=(x,)); (x,) = K._first_ngens(1)
>>> K.valuation(x).restriction(QQ)
Trivial valuation on Rational Field

simplify(f, error=None, force=False)
Return a simplified version of f.

Produce an element which differs from f by an element of valuation strictly greater than the valuation of f (or strictly greater than error if set.)

If force is not set, then expensive simplifications may be avoided.

EXAMPLES:

```python
sage: K.<x> = FunctionField(QQ)
sage: v = K.valuation(2)
sage: f = (x + 1)/(x - 1)

>>> from sage.all import *
>>> K = FunctionField(QQ, names=('x',)); (x,) = K._first_ngens(1)
>>> v = K.valuation(Integer(2))
>>> f = (x + Integer(1))/(x - Integer(1))

As the coefficients of this fraction are small, we do not simplify as this could be very costly in some cases:

```python
sage: v.simplify(f)
(x + 1)/(x - 1)

>>> from sage.all import *
>>> v.simplify(f)
(x + 1)/(x - 1)

However, simplification can be forced:

```python
sage: v.simplify(f, force=True)
3
Discrete Valuations and Discrete Pseudo-Valuations, Release 10.4

```
>>> from sage.all import *
>>> v.simplify(f, force=True)
3
```

```
uniformizer()

Return a uniformizing element for this valuation.

EXAMPLES:

```
sage: K.<x> = FunctionField(QQ)
sage: K.valuation(x).uniformizer()
x
```

```
value_group()

Return the value group of this valuation.

EXAMPLES:

```
sage: K.<x> = FunctionField(QQ)
sage: K.valuation(x).value_group()
Additive Abelian Group generated by 1
```
```
```
class sage.rings.function_field.valuation.InfiniteRationalFunctionFieldValuation(parent)

Bases: FunctionFieldMapedValuationRelative_base, RationalFunctionFieldValuation_base, ClassicalFunctionFieldvaluation_base

Valuation of the infinite place of a function field.

EXAMPLES:

```
sage: K.<x> = FunctionField(QQ)
sage: v = K.valuation(1/x)  # indirect doctest
```
```
```
class sage.rings.function_field.valuation.NonClassicalRationalFunctionFieldValuation(parent, base_valuation)

Bases: InducedRationalFunctionFieldValuation_base, RationalFunctionFieldvaluation_base

5.12. Discrete valuations on function fields 141
Valuation induced by a valuation on the underlying polynomial ring which is non-classical.

**EXAMPLES:**

```python
sage: K.<x> = FunctionField(QQ)
sage: v = GaussValuation(QQ['x'], QQ.valuation(2))
sage: w = K.valuation(v); w  # indirect doctest
2-adic valuation
```

```python
>>> from sage.all import *
>>> K = FunctionField(QQ, names=('x',)); (x,) = K._first_ngens(1)
>>> v = GaussValuation(QQ['x'], QQ.valuation(Integer(2)))
>>> w = K.valuation(v); w  # indirect doctest
2-adic valuation
```

```python
residue_ring()
```

Return the residue field of this valuation.

**EXAMPLES:**

```python
sage: K.<x> = FunctionField(QQ)
sage: v = valuations.GaussValuation(QQ['x'], QQ.valuation(2))
sage: w = K.valuation(v)
sage: w.residue_ring()
Rational function field in x over Finite Field of size 2
```

```python
>>> R.<x> = QQ[]
>>> vv = v.augmentation(x, 1)
>>> w = K.valuation(vv)
>>> w.residue_ring()
Rational function field in x over Finite Field of size 2
```

```python
sage: R.<y> = K[]
sage: L.<y> = K.extension(y^2 + 2*x)  # needs sage.rings.function_field
>>> w.extension(L).residue_ring()  # needs sage.rings.function_field
Function field in u2 defined by u2^2 + x
```

```python
>>> from sage.all import *
>>> K = FunctionField(QQ, names=('x',)); (x,) = K._first_ngens(1)
>>> v = valuations.GaussValuation(QQ['x'], QQ.valuation(Integer(2)))
>>> w = K.valuation(v)
>>> w.residue_ring()
Rational function field in x over Finite Field of size 2
```

```python
>>> R = QQ['x']; (x,) = R._first_ngens(1)
>>> vv = v.augmentation(x, Integer(1))
>>> w = K.valuation(vv)
>>> w.residue_ring()
Rational function field in x over Finite Field of size 2
```

```python
>>> R = K['y']; (y,) = R._first_ngens(1)
>>> L = K.extension(y**Integer(2) + Integer(2)*x, names=('y',)); (y,) = L._first_ngens(1)  # needs sage.rings.function_field
>>> w.extension(L).residue_ring()  # needs sage.rings.function_field
Function field in u2 defined by u2^2 + x
```
class sage.rings.function_field.valuation.RationalFunctionFieldMappedValuation (parent, base_valuation, to_base_valuation, from_base_valuation_domain)

Bases: FunctionFieldMappedValuationRelative_base, RationalFunctionFieldValuation_base

Valuation on a rational function field that is implemented after a map to an isomorphic rational function field.

EXAMPLES:

sage: K.<x> = FunctionField(QQ)
sage: R.<x> = QQ[]
sage: w = 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()]))); 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)

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

class sage.rings.function_field.valuation.RationalFunctionFieldValuation_base (parent)

Bases: FunctionFieldValuation_base

Base class for valuations on rational function fields.

element_with_valuation (s)

Return an element with valuation s.

EXAMPLES:

sage: # needs sage.rings.number_field
sage: x = polygen(ZZ, 'x')
sage: K.<a> = NumberField(x^3 + 6)
sage: v = K.valuation(2)
sage: R.<x> = K[]

(continues on next page)
sage: w = GaussValuation(R, v).augmentation(x, 1/123)
sage: K.<x> = FunctionField(K)
sage: w = w.extension(K)
sage: w.element_with_valuation(122/123)
2/x
sage: w.element_with_valuation(1)
2

>>> from sage.all import *
>>> # needs sage.rings.number_field
>>> x = polygen(ZZ, 'x')
>>> K = NumberField(x**Integer(3) + Integer(6), names=('a',)); (a,) = K._˓→first_ngens(1)
>>> v = K.valuation(Integer(2))
>>> R = K['x']; (x,) = R._first_ngens(1)
>>> w = GaussValuation(R, v).augmentation(x, Integer(1)/Integer(123))
>>> K = FunctionField(K, names=('x',)); (x,) = K._first_ngens(1)
>>> w = w.extension(K)
>>> w.element_with_valuation(Integer(122)/Integer(123))
2/x
>>> w.element_with_valuation(Integer(1))
2

5.13 \( p \)-adic Valuations on Number Fields and Their Subrings and Completions

EXAMPLES:

sage: ZZ.valuation(2)
2-adic valuation
sage: QQ.valuation(3)
3-adic valuation
sage: CyclotomicField(5).valuation(5)  # needs sage.rings.number_field
5-adic valuation
sage: GaussianIntegers().valuation(7)  # needs sage.rings.number_field
7-adic valuation
sage: Zp(11).valuation()
11-adic valuation

>>> from sage.all import *
>>> ZZ.valuation(Integer(2))
2-adic valuation
>>> QQ.valuation(Integer(3))
3-adic valuation
>>> CyclotomicField(Integer(5)).valuation(Integer(5))  # needs sage.rings.number_field
5-adic valuation
>>> GaussianIntegers().valuation(Integer(7))  # needs sage.rings.number_field
7-adic valuation
These valuations can then, e.g., be used to compute approximate factorizations in the completion of a ring:

```python
sage: v = ZZ.valuation(2)
sage: R.<x> = ZZ[]
sage: f = x^5 + x^4 + x^3 + x^2 + x - 1
sage: v.montes_factorization(f, required_precision=20)  # needs sage.geometry.polyhedron
(x + 676027) * (x^4 + 372550*x^3 + 464863*x^2 + 385052*x + 297869)
```

```python
>> from sage.all import *
>> v = ZZ.valuation(Integer(2))
>> R = ZZ['x']; (x,) = R._first_ngens(1)
>> f = x**Integer(5) + x**Integer(4) + x**Integer(3) + x**Integer(2) + x - Integer(1)
>> v.montes_factorization(f, required_precision=Integer(20))  # needs sage.geometry.polyhedron
(x + 676027) * (x^4 + 372550*x^3 + 464863*x^2 + 385052*x + 297869)
```

AUTHORS:

- Julian Rüth (2013-03-16): initial version

REFERENCES:

The theory used here was originally developed in [Mac1936I] and [Mac1936II]. An overview can also be found in Chapter 4 of [Rüt2014].

```python
class sage.rings.padics.padic_valuation.PadicValuationFactory
    Bases: UniqueFactory

    Create a prime-adic valuation on R.

    INPUT:

    - R – a subring of a number field or a subring of a local field in characteristic zero
    - prime – a prime that does not split, a discrete (pseudo-)valuation, a fractional ideal, or None (default: None)

    EXAMPLES:

    For integers and rational numbers, prime is just a prime of the integers:

```
Discrete Valuations and Discrete Pseudo-Valuations, Release 10.4

\begin{verbatim}
sage: valuations.pAdicValuation(Qp(2))
2-adic valuation
sage: valuations.pAdicValuation(Zp(2))
2-adic valuation

>>> from sage.all import *

>>> valuations.pAdicValuation(Qp(Integer(2)))
2-adic valuation

>>> valuations.pAdicValuation(Zp(Integer(2)))
2-adic valuation

But it must be specified in all other cases:

sage: valuations.pAdicValuation(ZZ)
Traceback (most recent call last):
    ...
ValueError: prime must be specified for this ring

>>> from sage.all import *

>>> valuations.pAdicValuation(ZZ)
Traceback (most recent call last):
    ...
ValueError: prime must be specified for this ring

It can sometimes be beneficial to define a number field extension as a quotient of a polynomial ring (since number field extensions always compute an absolute polynomial defining the extension which can be very costly):

\begin{verbatim}
sage: # needs sage.rings.number_field
sage: R.<x> = QQ[]
sage: K.<a> = NumberField(x^2 + 1)
sage: R.<x> = K[]
sage: L.<b> = R.quo(x^2 + a)
sage: valuations.pAdicValuation(L, 2)
2-adic valuation

>>> from sage.all import *

>>> # needs sage.rings.number_field

>>> R = QQ['x']; (x,) = R._first_ngens(1)

>>> K = NumberField(x**Integer(2) + Integer(1), names=('a',)); (a,) = K._first_ngens(1)

>>> R = K['x']; (x,) = R._first_ngens(1)

>>> L = R.quo(x**Integer(2) + a, names=('b',)); (b,) = L._first_ngens(1)

>>> valuations.pAdicValuation(L, Integer(2))
2-adic valuation
\end{verbatim}

See also:

NumberField_generic.valuation(), Order.valuation(), pAdicGeneric.valuation(), RationalField.valuation(), IntegerRing_class.valuation()

create_key_and_extra_args(R, prime=None, approximants=None)

Create a unique key identifying the valuation of \texttt{R} with respect to \texttt{prime}.

EXAMPLES:
Discrete Valuations and Discrete Pseudo-Valuations, Release 10.4

sage: QQ.valuation(2)  # indirect doctest
2-adic valuation

```python
>>> from sage.all import *
>>> QQ.valuation(Integer(2))  # indirect doctest
2-adic valuation
```

**create_key_and_extra_args_for_number_field** \((R, \text{prime}, \text{approximants})\)
Create a unique key identifying the valuation of \(R\) with respect to \(\text{prime}\).

**EXAMPLES:**

```python
sage: GaussianIntegers().valuation(2)  # indirect doctest  # needs sage.rings.number_field
2-adic valuation
```

```python
>>> from sage.all import *
>>> GaussianIntegers().valuation(Integer(2))  # indirect doctest  # needs sage.rings.number_field
2-adic valuation
```

**create_key_and_extra_args_for_number_field_from_ideal** \((R, I, \text{prime})\)
Create a unique key identifying the valuation of \(R\) with respect to \(I\).

**Note:** \(\text{prime}\), the original parameter that was passed to **create_key_and_extra_args()**, is only used to provide more meaningful error messages.

**EXAMPLES:**

```python
sage: # needs sage.rings.number_field
sage: GaussianIntegers().valuation(GaussianIntegers().number_field().fractional_ideal(2))  # indirect doctest
2-adic valuation
```

```python
>>> from sage.all import *
>>> # needs sage.rings.number_field
>>> GaussianIntegers().valuation(GaussianIntegers().number_field().fractional_ideal(Integer(2)))  # indirect doctest
2-adic valuation
```

**create_key_and_extra_args_for_number_field_from_valuation** \((R, v, \text{prime}, \text{approximants})\)
Create a unique key identifying the valuation of \(R\) with respect to \(v\).

**Note:** \(\text{prime}\), the original parameter that was passed to **create_key_and_extra_args()**, is only used to provide more meaningful error messages.

**EXAMPLES:**

```python
sage: # needs sage.rings.number_field
sage: GaussianIntegers().valuation(ZZ.valuation(2))  # indirect doctest  # needs sage.rings.number_field
2-adic valuation
```

5.13. \(p\)-adic Valuations on Number Fields and Their Subrings and Completions

147
```python
>>> from sage.all import *
>>> GaussianIntegers().valuation(ZZ.valuation(Integer(2)))  # indirect...
˓→doctest  # needs sage.rings.number_field
2-adic valuation
```

**create_key_for_integers**(*R, prime*)

Create a unique key identifying the valuation of *R* with respect to *prime*.

**EXAMPLES:**

```python
sage: QQ.valuation(2)  # indirect doctest
2-adic valuation
```

```python
>>> from sage.all import *
>>> QQ.valuation(Integer(2))  # indirect doctest
2-adic valuation
```

**create_key_for_local_ring**(*R, prime*)

Create a unique key identifying the valuation of *R* with respect to *prime*.

**EXAMPLES:**

```python
sage: Qp(2).valuation()  # indirect doctest
2-adic valuation
```

```python
>>> from sage.all import *
>>> Qp(Integer(2)).valuation()  # indirect doctest
2-adic valuation
```

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

Create a *p*-adic valuation from *key*.

**EXAMPLES:**

```python
sage: ZZ.valuation(5)  # indirect doctest
5-adic valuation
```

```python
>>> from sage.all import *
>>> ZZ.valuation(Integer(5))  # indirect doctest
5-adic valuation
```

**class** `sage.rings.padics.padic_valuation.pAdicFromLimitValuation` *(parent, approximant, G, approximants)*

A *p*-adic valuation on a number field or a subring thereof, i.e., a valuation that extends the *p*-adic valuation on the integers.

**EXAMPLES:**

```python
sage: v = GaussianIntegers().valuation(3); v  # indirect doctest
˓→needs sage.rings.number_field
3-adic valuation
```
Discrete Valuations and Discrete Pseudo-Valuations, Release 10.4

```python
>>> from sage.all import *
>>> v = GaussianIntegers().valuation(Integer(3)); v
... # needs sage.rings.number_field
3-adic valuation
```

### extensions (ring)

Return the extensions of this valuation to ring.

**EXAMPLES:**

```python
sage: v = GaussianIntegers().valuation(3)  # needs sage.rings.number_field
sage: v.extensions(v.domain().fraction_field())  # needs sage.rings.number_field
[3-adic valuation]
```

```python
>>> from sage.all import *
>>> v = GaussianIntegers().valuation(Integer(3))  # needs sage.rings.number_field
>>> v.extensions(v.domain().fraction_field())  # needs sage.rings.number_field
[3-adic valuation]
```

### class sage.rings.padics.padic_valuation.pAdicValuation_base (parent, p)

**Bases:** DiscreteValuation

Abstract base class for $p$-adic valuations.

**INPUT:**

- `ring` — an integral domain
- `p` — a rational prime over which this valuation lies, not necessarily a uniformizer for the valuation

**EXAMPLES:**

```python
sage: ZZ.valuation(3)
3-adic valuation
sage: QQ.valuation(5)
5-adic valuation
```

For `$p$'-adic rings, `$p$` has to match the `$p$` of the ring. ::

```python
sage: v = valuations.pAdicValuation(Zp(3), 2); v
Traceback (most recent call last):
... ValueError: prime must be an element of positive valuation
```

### change_domain (ring)

Change the domain of this valuation to ring if possible.

**EXAMPLES:**

```python
sage: v = ZZ.valuation(2)
sage: v.change_domain(QQ).domain()
Rational Field
```
extensions (ring)

Return the extensions of this valuation to ring.

EXAMPLES:

```python
>>> from sage.all import *
>>> v = ZZ.valuation(Integer(2))
>>> v.change_domain(QQ).domain()
Rational Field
```

is_totally_ramified (G, include_steps=False, assume_squarefree=False)

Return whether G defines a single totally ramified extension of the completion of the domain of this valuation.

INPUT:

- G – a monic squarefree polynomial over the domain of this valuation
- include_steps – a boolean (default: False); where to include the valuations produced during the process of checking whether G is totally ramified in the return value
- assume_squarefree – a boolean (default: False); whether to assume that G is square-free over the completion of the domain of this valuation. Setting this to True can significantly improve the performance.

ALGORITHM:

This is a simplified version of `sage.rings.valuation.valuation.DiscreteValuation.mac_lane_approximants()`.

EXAMPLES:

```python
sage: # needs sage.libsntl
sage: k = Qp(5,4)
sage: v = k.valuation()
sage: R.<x> = k[]
sage: G = x^2 + 1
sage: v.is_totally_ramified(G)  # needs sage.geometry.polyhedron
False
sage: G = x + 1
sage: v.is_totally_ramified(G)
True
sage: G = x^2 + 2
sage: v.is_totally_ramified(G)
False
sage: G = x^2 + 5
sage: v.is_totally_ramified(G)  # needs sage.geometry.polyhedron
```

(continues on next page)
We consider an extension as totally ramified if its ramification index matches the degree. Hence, a trivial extension is totally ramified:

```
sage: R.<x> = QQ[]  
sage: v = QQ.valuation(2)  
sage: v.is_totally_ramified(x)  
True
```

We consider an extension as totally ramified if its ramification index matches the degree. Hence, a trivial extension is totally ramified:

```
sage: R.<x> = QQ[]  
sage: v = QQ.valuation(2)  
sage: v.is_totally_ramified(x)  
True
```

We consider an extension as totally ramified if its ramification index matches the degree. Hence, a trivial extension is totally ramified:

```
sage: R.<x> = QQ[]  
sage: v = QQ.valuation(2)  
sage: v.is_totally_ramified(x)  
True
```

The `is_unramified` function returns whether G defines a single unramified extension of the completion of the domain of this valuation.

```
>>> from sage.all import *  
>>> R = QQ['x']; (x,) = R._first_ngens(1)  
>>> v = QQ.valuation(Integer(2))  
>>> v.is_totally_ramified(x)  
True
```

```
>>> from sage.all import *  
>>> R = QQ['x']; (x,) = R._first_ngens(1)  
>>> v = QQ.valuation(Integer(2))  
>>> v.is_totally_ramified(x)  
True
```

```is_unramified(G, include_steps=False, assume_squarefree=False)```

Return whether G defines a single unramified extension of the completion of the domain of this valuation.

**INPUT:**

- G – a monic squarefree polynomial over the domain of this valuation
- include_steps – a boolean (default: False); whether to include the approximate valuations that were used to determine the result in the return value.
- assume_squarefree – a boolean (default: False); whether to assume that G is square-free over the completion of the domain of this valuation. Setting this to True can significantly improve the performance.
EXAMPLES:

We consider an extension as unramified if its ramification index is 1. Hence, a trivial extension is unramified:

```
sage: R.<x> = QQ[]
sage: v = QQ.valuation(2)
sage: v.is_unramified(x)
True
```

If $G$ remains irreducible in reduction, then it defines an unramified extension:

```
sage: v.is_unramified(x^2 + x + 1)
True
```

However, even if $G$ factors, it might define an unramified extension:

```
sage: v.is_unramified(x^2 + 2*x + 4)  # needs sage.geometry.polyhedron
True
```

**lift** $(x)$

Lift $x$ from the residue field to the domain of this valuation.

**INPUT:**

- $x$ – an element of the residue_field()

**EXAMPLES:**

```
sage: v = ZZ.valuation(3)
sage: xbar = v.reduce(4)
sage: v.lift(xbar)
1
```

**$p$**

Return the $p$ of this $p$-adic valuation.

**EXAMPLES:**

```
```
Discrete Valuations and Discrete Pseudo-Valuations, Release 10.4

\(\text{sage: GaussianIntegers().valuation(2).p()}\)  
\(#\text{needs sage.rings.number_field}\)  
\(\text{2}\)

```python
from sage.all import *
GaussianIntegers().valuation(Integer(2)).p()  
# needs sage.rings.number_field
2
```

**reduce** \((x)\)
Reduce \(x\) modulo the ideal of elements of positive valuation.

**INPUT:**
- \(x\) – an element in the domain of this valuation

**OUTPUT:**
An element of the `residue_field()`.

**EXAMPLES:**
```python
sage: v = ZZ.valuation(3)
sage: v.reduce(4)
1
```

```python
from sage.all import *
from sage.all import *
v = ZZ.valuation(Integer(3))
v.reduce(Integer(4))
1
```

**restriction** \((\text{ring})\)
Return the restriction of this valuation to \(\text{ring}\).

**EXAMPLES:**
```python
sage: v = GaussianIntegers().valuation(2)  
# needs sage.rings.number_field
sage: v.restriction(ZZ)  
# needs sage.rings.number_field
2-adic valuation
```

```python
from sage.all import *
from sage.all import *
from sage.all import *
v = GaussianIntegers().valuation(Integer(2))  
# needs sage.rings.number_field
>>> v.restriction(ZZ)  
# needs sage.rings.number_field
2-adic valuation
```

**value_semigroup()**
Return the value semigroup of this valuation.

**EXAMPLES:**
```python
sage: v = GaussianIntegers().valuation(2)  
# needs sage.rings.number_field
sage: v.value_semigroup()  
# needs sage.rings.number_field
```

(continues on next page)
Additive Abelian Semigroup generated by 1/2

```python
>>> from sage.all import *

>>> v = GaussianIntegers().valuation(Integer(2))
# needs sage.rings.number_field

>>> v.value_semigroup()
# needs sage.rings.number_field
```

```python
class sage.rings.padics.padic_valuation.pAdicValuation_int (parent, p)
```

Bases: `pAdicValuation_base`

A $p$-adic valuation on the integers or the rationals.

EXAMPLES:

```python
sage: v = ZZ.valuation(3); v
3-adic valuation

sage: from sage.all import *

sage: v = ZZ.valuation(Integer(3)); v
3-adic valuation

inverse (x, precision)

Return an approximate inverse of x.

The element returned is such that the product differs from 1 by an element of valuation at least `precision`.

INPUT:

- x – an element in the domain of this valuation
- precision – a rational or infinity

EXAMPLES:

```python
sage: v = ZZ.valuation(2)
sage: x = 3
sage: y = v.inverse(3, 2); y
3
sage: x*y - 1
8

sage: from sage.all import *

sage: v = ZZ.valuation(Integer(2))

sage: x = Integer(3)

sage: y = v.inverse(Integer(3), Integer(2)); y
3

sage: x*y - Integer(1)
8
```

This might not be possible for elements of positive valuation:

```python
sage: v.inverse(2, 2)
Traceback (most recent call last):
...
ValueError: element has no approximate inverse in this ring
```
>>> from sage.all import *
>>> v.inverse(Integer(2), Integer(2))
Traceback (most recent call last):
  ...
ValueError: element has no approximate inverse in this ring

Unless the precision is very small:

sage: v.inverse(2, 0)
1

residue_ring()
Return the residue field of this valuation.

EXAMPLES:

sage: v = ZZ.valuation(3)
sage: v.residue_ring()
Finite Field of size 3

sage: v = ZZ.valuation(Integer(3))
>>> v.residue_ring()
Finite Field of size 3

simplify(x, error=None, force=False, size_heuristic_bound=32)
Return a simplified version of x.

Produce an element which differs from x by an element of valuation strictly greater than the valuation of x (or strictly greater than error if set.)

INPUT:

• x – an element in the domain of this valuation
• error – a rational, infinity, or None (default: None), the error allowed to introduce through the simplification
• force – ignored
• size_heuristic_bound – when force is not set, the expected factor by which the x need to shrink to perform an actual simplification (default: 32)

EXAMPLES:

sage: v = ZZ.valuation(2)
sage: v.simplify(6, force=True)
2
sage: v.simplify(6, error=0, force=True)
0

>>> from sage.all import *
>>> v = ZZ.valuation(Integer(2))
>>> v.simplify(Integer(6), force=True)
(continues on next page)
In this example, the usual rational reconstruction misses a good answer for some moduli (because the absolute value of the numerator is not bounded by the square root of the modulus):

```python
sage: v = QQ.valuation(2)
sage: v.simplify(110406, error=16, force=True)
562/19
sage: Qp(2, 16)(110406).rational_reconstruction()
Traceback (most recent call last):
... ArithmeticError: rational reconstruction of 55203 (mod 65536) does not exist
```

### uniformizer()

Return a uniformizer of this $p$-adic valuation, i.e., $p$ as an element of the domain.

**EXAMPLES:**

```python
sage: v = ZZ.valuation(3)
sage: v.uniformizer()
3
```
element_with_valuation($v$)

Return an element of valuation $v$.

INPUT:

- $v$ – an element of the $\text{pAdicValuation\_base.value\_semigroup()}$ of this valuation

EXAMPLES:

```python
sage: R = Zp(3)
sage: v = R.valuation()
sage: v.element_with_valuation(3)
3^3 + O(3^23)

sage: # needs sage.libs.ntl
sage: K = Qp(3)
sage: R.<y> = K[]
sage: L.<y> = K.extension(y^2 + 3*y + 3)
sage: L.valuation().element_with_valuation(3/2)
y^3 + O(y^43)
```

lift($x$)

Lift $x$ from the $\text{residue\_field()}$ to the domain of this valuation.

INPUT:

- $x$ – an element of the residue field of this valuation

EXAMPLES:

```python
>>> from sage.all import *
>>> R = Zp(Integer(3))
>>> v = R.valuation()
>>> xbar = v.reduce(R(4))
>>> v.lift(xbar)
1 + O(3^20)
```

reduce($x$)

Reduce $x$ modulo the ideal of elements of positive valuation.

```
INPUT:

- $x$ – an element of the domain of this valuation

OUTPUT:

An element of the $\text{residue_field()}$.

EXAMPLES:

```sage
sage: R = Zp(3)
sage: Zp(3).valuation().reduce(R(4))
1
```

```sage
>>> from sage.all import *
>>> R = Zp(Integer(3))
>>> Zp(Integer(3)).valuation().reduce(R(Integer(4)))
1
```

```sage
residue_ring()

Return the residue field of this valuation.

EXAMPLES:

```sage
sage: Qq(9, names = 'a').valuation().residue_ring()  # needs sage.libs.ntl
Finite Field in a0 of size 3^2
```

```sage
>>> from sage.all import *
>>> Qq(Integer(9), names = 'a').valuation().residue_ring()  # needs sage.libs.ntl
Finite Field in a0 of size 3^2
```

```sage
shift (x, s)

Shift $x$ in its expansion with respect to $\text{uniformizer()}$ by $s$ “digits”.

For non-negative $s$, this just returns $x$ multiplied by a power of the uniformizer $\pi$.

For negative $s$, it does the same but when not over a field, it drops coefficients in the $\pi$-adic expansion which have negative valuation.

EXAMPLES:

```sage
sage: R = ZpCA(2)
sage: v = R.valuation()
sage: v.shift(R.one(), 1)
2 + O(2^20)
sage: v.shift(R.one(), -1)
O(2^19)
```

```sage
# needs sage.libs.ntl sage.rings.padics
sage: S.<y> = R[]
sage: S.<y> = R.extension(y^3 - 2)
sage: v = S.valuation()
sage: v.shift(1, 5)
y^5 + O(y^60)
```

```sage
>>> from sage.all import *
>>> R = ZpCA(Integer(2))
```
Discrete Valuations and Discrete Pseudo-Valuations, Release 10.4

Discrete Valuations and Discrete Pseudo-Valuations, Release 10.4

simplify \(x, \text{error}=\text{None}, \text{force}=\text{False}\)

Return a simplified version of \(x\).

Produce an element which differs from \(x\) by an element of valuation strictly greater than the valuation of \(x\) (or strictly greater than \(\text{error}\) if set.)

**INPUT:**

- \(x\) – an element in the domain of this valuation
- \(\text{error}\) – a rational, infinity, or None (default: None), the error allowed to introduce through the simplification
- \(\text{force}\) – ignored

**EXAMPLES:**

\[
\begin{align*}
\text{sage}: & \quad R = \mathbb{Z}_p(2) \\
\text{sage}: & \quad v = R.\text{valuation}() \\
\text{sage}: & \quad v.\text{simplify}(6) \\
& \quad 2 + 0(2^{21})
\end{align*}
\]

\[
\begin{align*}
\text{sage}: & \quad v.\text{simplify}(6, \text{error}=0) \\
& \quad 0
\end{align*}
\]

\[
\begin{align*}
\text{from sage.all import } & \ast \\
\text{from sage.all import } & \ast \\
\text{sage}: & \quad R = \mathbb{Z}_p(\text{Integer}(2)) \\
\text{sage}: & \quad v = R.\text{valuation}() \\
\text{sage}: & \quad v.\text{simplify}(\text{Integer}(6)) \\
& \quad 2 + 0(2^{21})
\end{align*}
\]

\[
\begin{align*}
\text{sage}: & \quad v.\text{simplify}(\text{Integer}(6), \text{error}=\text{Integer}(0)) \\
& \quad 0
\end{align*}
\]

uniformizer()

Return a uniformizer of this valuation.

**EXAMPLES:**

\[
\begin{align*}
\text{sage}: & \quad v = \mathbb{Z}_p(3).\text{valuation}() \\
\text{sage}: & \quad v.\text{uniformizer}() \\
& \quad 3 + 0(3^{21})
\end{align*}
\]

\[
\begin{align*}
\text{from sage.all import } & \ast \\
\text{v} & \quad = \mathbb{Z}_p(\text{Integer}(3)).\text{valuation}()
\end{align*}
\]
3 + O(3^{21})
INDICES AND TABLES

- Index
- Module Index
- Search Page
sage.rings.function_field.valuation, 129
sage.rings.padics.padic_valuation, 144
sage.rings.valuation.augmented_valuation, 82
sage.rings.valuation.developing_valuation, 60
sage.rings.valuation.gauss_valuation, 47
sage.rings.valuation.inductive_valuation, 64
sage.rings.valuation.limit_valuation, 110
sage.rings.valuation.mapped_valuation, 119
sage.rings.valuation.scaled_valuation, 126
sage.rings.valuation.trivial_valuation, 42
sage.rings.valuation.valuation, 18
sage.rings.valuation.valuation_space, 28
sage.rings.valuation.value_group, 13
INDEX

A
augmentation() (sage.rings.valuation.inductivevaluation.NonFinalInductiveValuation method), 72
augmentation_chain() (sage.rings.valuation.augmentedvaluation.AugmentedValuation_base method), 85
augmentation_chain() (sage.rings.valuation.gaussvaluation.GaussValuation_generic method), 50
AugmentedValuation_base (class in sage.rings.valuation.augmentedvaluation), 66
AugmentedValuationFactory (class in sage.rings.valuation.augmentedvaluation), 83

C
change_domain() (sage.rings.padic.padics.pAdicValuation_base method), 149
change_domain() (sage.rings.valuation.augmentedvaluation.AugmentedValuation_base method), 85
change_domain() (sage.rings.valuation.gaussvaluation.GaussValuation_generic method), 50
change_domain() (sage.rings.valuation.inductivevaluation.InfiniteInductiveValuation method), 71
change_domain() (sage.rings.valuation.valuation_space.DiscretePseudoValuationSpace.ElementMethods method), 31
ClassicalFunctionFieldValuation_base (class in sage.rings.function_field.valuation), 130
coefficients() (sage.rings.valuation.developingvaluation.DevelopingValuation method), 61
create_key() (sage.rings.valuation.augmentedvaluation.AugmentedValuationFactory method), 83
create_key() (sage.rings.valuation.limitvaluation.LimitValuationFactory method), 112
create_key() (sage.rings.valuation.scaledvaluation.ScaledValuationFactory method), 127
create_key() (sage.rings.valuation.trivialvaluation.TrivialValuationFactory method), 46
create_key_and_extra_args() (sage.rings.function_field.valuation.FunctionFieldValuationFactory method), 137
create_key_and_extra_args() (sage.rings.padic.padics.pAdicValuationFactory method), 146
create_key_and_extra_args_for_number_field() (sage.rings.padic.padics.pAdicValuationFactory method), 147
create_key_and_extra_args_for_number_field_from_ideal() (sage.rings.padic.padics.pAdicValuationFactory method), 147
create_key_and_extra_args_for_number_field_from_valuation() (sage.rings.padic.padics.pAdicValuationFactory method), 147
create_key_and_extra_args_from_place() (sage.rings.function_field.valuation.FunctionFieldValuationFactory method), 138
create_key_and_extra_args_from_valuation() (sage.rings.function_field.valuation.FunctionFieldValuationFactory method), 138
create_key_and_extra_args_from_valuation_on_isomorphic_field() (sage.rings.function_field.valuation.FunctionFieldValuationFactory method), 138
create_key_for_integers() (sage.rings.padic.padics.pAdicValuationFactory method), 148
create_key_for_local_ring() (sage.rings.padic.padics.pAdicValuationFactory method), 148
create_object() (sage.rings.function_field.valuation.FunctionFieldValuationFactory method), 138
create_object() (sage.rings.padic.padics.pAdicValuationFactory method), 148
<table>
<thead>
<tr>
<th>Function/Class/Method</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>element_with_valuation() (sage.rings.valuation.gaussvaluation.GaussValuation_generic.method)</td>
<td>50</td>
</tr>
<tr>
<td>element_with_valuation() (sage.rings.valuation.inductivevaluation.InductiveValuation.method)</td>
<td>66</td>
</tr>
<tr>
<td>element_with_valuation() (sage.rings.valuation.limitvaluation.LimitValuationFactory.method)</td>
<td>114</td>
</tr>
<tr>
<td>element_with_valuation() (sage.rings.valuation.mappedvaluation.MappedValuation_base.method)</td>
<td>123</td>
</tr>
<tr>
<td>equivalence_decomposition() (sage.rings.valuation.inductivevaluation.NonFinalInductiveValuation.method)</td>
<td>73</td>
</tr>
<tr>
<td>equivalence_reciprocal() (sage.rings.valuation.inductivevaluation.InductiveValuation.method)</td>
<td>67</td>
</tr>
<tr>
<td>equivalence_unit() (sage.rings.valuation.augmentedvaluation.AugmentedValuation_base.method)</td>
<td>87</td>
</tr>
<tr>
<td>equivalence_unit() (sage.rings.valuation.gaussvaluation.GaussValuation_generic.method)</td>
<td>51</td>
</tr>
<tr>
<td>extension() (sage.rings.valuation.valuation_space.DiscretePseudoValuationSpace.ElementMethods.method)</td>
<td>68</td>
</tr>
<tr>
<td>extension() (sage.rings.function_field.valuation.DiscreteFunctionFieldValuation_base.method)</td>
<td>130</td>
</tr>
<tr>
<td>extension() (sage.rings.function_field.valuation.InducedRationalFunctionFieldValuation_base.method)</td>
<td>138</td>
</tr>
<tr>
<td>extension() (sage.rings.padics.padicvaluation.pAdicFromLimitValuation.method)</td>
<td>149</td>
</tr>
<tr>
<td>extension() (sage.rings.padics.padicvaluation.pAdicValuation_base.method)</td>
<td>150</td>
</tr>
<tr>
<td>extension() (sage.rings.valuation.augmentedvaluation.AugmentedValuation_base.method)</td>
<td>88</td>
</tr>
<tr>
<td>extension() (sage.rings.valuation.gaussvaluation.GaussValuation_generic.method)</td>
<td>51</td>
</tr>
<tr>
<td>extension() (sage.rings.valuation.inductivevaluation.FiniteInductiveValuation.method)</td>
<td>65</td>
</tr>
<tr>
<td>extension() (sage.rings.valuation.limitvaluation.MacLaneLimitValuation.method)</td>
<td>114</td>
</tr>
<tr>
<td>extension() (sage.rings.valuation.scaledvaluation.ScaledValuation_generic.method)</td>
<td>48</td>
</tr>
<tr>
<td>extension() (sage.rings.valuation.scaledvaluation.ScaledValuation_generic.method)</td>
<td>127</td>
</tr>
</tbody>
</table>

**D**

- denominator() (sage.rings.valuation.value_group.DiscreteValueGroup.method) | 14   |
- DevelopingValuation (class in sage.rings.valuation.developing_valuation) | 60   |
- DiscreteFunctionFieldValuation_base (class in sage.rings.function_field.valuation) | 130  |
- DiscretePseudoValuation (class in sage.rings.valuation.valuation) | 19   |
- DiscretePseudoValuationSpace (class in sage.rings.valuation.valuation_space) | 29   |
- DiscretePseudoValuationSpace.ElementMethods (class in sage.rings.valuation.valuation_space) | 30   |
- DiscreteValuation (class in sage.rings.valuation.valuation) | 20   |
- DiscreteValuationCodomain (class in sage.rings.valuation.value_group) | 13   |
- DiscreteValueGroup (class in sage.rings.valuation.value_group) | 13   |
- DiscreteValueSemigroup (class in sage.rings.valuation.value_group) | 16   |

**E**

- E() (sage.rings.valuation.augmentedvaluation.AugmentedValuation_base.method) | 84   |
- E() (sage.rings.valuation.gaussvaluation.GaussValuation_generic.method) | 49   |
- E() (sage.rings.valuation.inductivevaluation.InductiveValuation.method) | 65   |
- effective_degree() (sage.rings.valuation.developingvaluation.DevelopingValuationFactory.method) | 61   |
- element_with_valuation() (sage.rings.function_field.valuation.InducedRationalFunctionFieldValuation_base.method) | 143  |
- element_with_valuation() (sage.rings.padics.padicvaluation.pAdicValuation_padicmethod) | 156  |
- element_with_valuation() (sage.rings.valuation.augmentedvaluation.AugmentedValuation_base.method) | 86   |
Index 

169
reduce() (sage.rings.valuation.gauss_valuation.GaussValuation_generic method), 55
reduce() (sage.rings.valuation.limit_valuation.LimitValuation_generic method), 113
reduce() (sage.rings.valuation.mapped_valuation.MappedValuation_base method), 124
reduce() (sage.rings.valuation.scaled_valuation.ScaledValuation_generic method), 128
reduce() (sage.rings.valuation.trivial_valuation.TrivialDiscretePseudoValuation method), 43
reduce() (sage.rings.valuation.trivial_valuation.TrivialDiscreteValuation method), 45
reduce() (sage.rings.valuation.valuation_space.DiscretePseudoValuationSpace.ElementMethods method), 34
residue_ring() (sage.rings.function_field.valuation.InducedRationalFunctionFieldValuation_base method), 139
residue_ring() (sage.rings.function_field.valuation.NonClassicalRationalFunctionFieldValuation_base method), 142
residue_ring() (sage.rings.padic.padic_valuation.pAdicValuation_int method), 155
residue_ring() (sage.rings.padic.padic_valuation.pAdicValuation_padic method), 158
residue_ring() (sage.rings.valuation.augmented_valuation.FinalAugmentedValuation method), 94
residue_ring() (sage.rings.valuation.augmented_valuation.NonFinalAugmentedValuation method), 108
residue_ring() (sage.rings.valuation.gauss_valuation.GaussValuation_generic method), 56
residue_ring() (sage.rings.valuation.limit_valuation.MacLaneLimitValuation method), 116
residue_ring() (sage.rings.valuation.mapped_valuation.MappedValuation_base method), 125
residue_ring() (sage.rings.valuation.scaled_valuation.ScaledValuation_generic method), 128
residue_ring() (sage.rings.valuation.trivial_valuation.TrivialDiscretePseudoValuation method), 43
residue_ring() (sage.rings.valuation.trivial_valuation.TrivialDiscreteValuation method), 45
restriction() (sage.rings.function_field.valuation.InducedRationalFunctionFieldValuation_base method), 135
restriction() (sage.rings.padics.padic_valuation.pAdicValuation_base method), 153
restriction() (sage.rings.valuation.augmented_valuation.AugmentedValuation_base method), 91
restriction() (sage.rings.valuation.gauss_valuation.GaussValuation_generic method), 56
restriction() (sage.rings.valuation.limit_valuation.MacLaneLimitValuation method), 117
restriction() (sage.rings.valuation.mapped_valuation.FiniteExtensionFromInfiniteValuation method), 121
restriction() (sage.rings.valuation.scaled_valuation.ScaledValuation_generic method), 128
scale() (sage.rings.function_field.valuation.FunctionFieldMappedValuationRelative_base method), 134
scale() (sage.rings.function_field.valuation.FunctionFieldMappedValuation_base method), 136
scale() (sage.rings.valuation.augmented_valuation.AugmentedValuation_base method), 91
scale() (sage.rings.valuation.gauss_valuation.GaussValuation_generic method), 56
scale() (sage.rings.valuation.valuation_space.DiscretePseudoValuationSpace.ElementMethods method), 36
ScaleAction (class in sage.rings.valuation.valuation_space), 41
ScaledValuationFactory (class in sage.rings.valuation.scaled_valuation), 127
SeparatedValuationFactory (class in sage.rings.valuation.scaled_valuation), 127
separating_element() (sage.rings.valuation.valuation_space.DiscretePseudoValuationSpace.ElementMethods method), 38
shift() (sage.rings.padics.padicvaluation.pAdicValuation_padic method), 158
simplify() (sage.rings.function_field.valuation.InducedRationalFunctionFieldValuation_base method), 140
simplify() (sage.rings.padics.padicvaluation.pAdicValuation_int method), 156
simplify() (sage.rings.padics.padicvaluation.pAdicValuation_padic method), 159
simplify() (sage.rings.valuation.augmented_valuation.FiniteAugmentedValuation method), 96
simplify() (sage.rings.valuation.augmented_valuation.InfiniteAugmentedValuation method), 101
simplify() (sage.rings.valuation.gauss_valuation.GaussValuation_generic method), 57
simplify() (sage.rings.valuation.limit_valuation.MacLaneLimitValuation method), 117
simplify() (sage.rings.valuation.mapped_valuation.FiniteExtensionFromInfiniteValuation method), 121
simplify() (sage.rings.valuation.mapped_valuation.MappedValuation_base method), 125
some_elements() (sage.rings.valuation.value_group.DiscreteValuGroup method), 16
some_elements() (sage.rings.valuation.value_group.DiscreteValueGroupSemigroup method), 18

T
TrivialDiscretePseudoValuation (class in sage.rings.valuation.trivial_valuation), 42
TrivialDiscretePseudoValuation_base (class in sage.rings.valuation.trivial_valuation), 43
TrivialDiscreteValuation (class in sage.rings.valuation.trivial_valuation), 44
TrivialValuationFactory (class in sage.rings.valuation.trivial_valuation), 46

U
uniformizer() (sage.rings.function_field.valuation.InducedRationalFunctionFieldValuation_base method), 141
uniformizer() (sage.rings.padics.padicvaluation.pAdicValuation_int method), 156
uniformizer() (sage.rings.padics.padicvaluation.pAdicValuation_padic method), 159
uniformizer() (sage.rings.valuation.augmented_valuation.AugmentedValuation_base method), 91
uniformizer() (sage.rings.valuation.gauss_valuation.GaussValuation_generic method), 57
uniformizer() (sage.rings.valuation.limit_valuation.MacLaneLimitValuation method), 118
uniformizer() (sage.rings.valuation.mapped_valuation.MappedValuation_base method), 126
uniformizer() (sage.rings.valuation.scaled_valuation.ScaledValuation_generic method), 128
uniformizer() (sage.rings.valuation.trivial_valuation.TrivialDiscretePseudoValuation_base method), 44
upper_bound() (sage.rings.valuation.augmented_valuation.FiniteAugmentedValuation method), 98
upper_bound() (sage.rings.valuation.augmented_valuation.InfiniteAugmentedValuation method), 101
upper_bound() (sage.rings.valuation.gauss_valuation.GaussValuation_generic method), 58
upper_bound() (sage.rings.valuation.limit_valuation.MacLaneLimitValuation method), 118
upper_bound() (sage.rings.valuation.mapped_valuation.FiniteExtensionFromInfiniteValuation method), 122
upper_bound() (sage.rings.valuation.valuation_space.DiscretePseudoValuationSpace.ElementMethods method), 40

V
valuations() (sage.rings.valuation.augmented_valuation.FiniteAugmentedValuation method), 98
valuations() (sage.rings.valuation.augmented_valuation.InfiniteAugmentedValuation method), 102
valuations() (sage.rings.valuation.developing_valuation.DevelopingValuation method), 63
valuations() (sage.rings.valuation.gauss_valuation.GaussValuation_generic method), 58
value_group() (sage.rings.function_field.valuation.InducedRationalFunctionFieldValuation_base method), 141
value_group() (sage.rings.valuation.augmented_valuation.FiniteAugmentedValuation method), 99
value_group() (sage.rings.valuation.augmented_valuation.InfiniteAugmentedValuation method), 102
value_group() (sage.rings.valuation.gauss_valuation.GaussValuation_generic method), 59
value_group() (sage.rings.valuation.trivial_valuation.TrivialDiscretePseudoValuation method), 43
value_group() (sage.rings.valuation.trivial_valuation.TrivialDiscreteValuation method), 46
value_group() (sage.rings.valuation.valuation_space.DiscretePseudoValuationSpace.ElementMethods method), 40
value_semigroup() (sage.rings.padics.padic_valuation.pAdicValuation_base method), 153
value_semigroup() (sage.rings.valuation.augmented_valuation.FiniteAugmentedValuation method), 99
value_semigroup() (sage.rings.valuation.augmented_valuation.InfiniteAugmentedValuation method), 103
value_semigroup() (sage.rings.valuation.gauss_valuation.GaussValuation_generic method), 59
value_semigroup() (sage.rings.valuation.limit_valuation.MacLaneLimitValuation method), 119
value_semigroup() (sage.rings.valuation.scaled_valuation.ScaledValuation_generic method), 129