Discrete Valuations and Discrete Pseudo-Valuations

Release 10.1

The Sage Development Team

Aug 21, 2023
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1  High-Level Interface</td>
<td>1</td>
</tr>
<tr>
<td>2  Low-Level Interface</td>
<td>3</td>
</tr>
<tr>
<td>3  Mac Lane Approximants</td>
<td>5</td>
</tr>
<tr>
<td>4  References</td>
<td>7</td>
</tr>
<tr>
<td>5  More Details</td>
<td>9</td>
</tr>
<tr>
<td>6  Indices and Tables</td>
<td>101</td>
</tr>
<tr>
<td>Python Module Index</td>
<td>103</td>
</tr>
<tr>
<td>Index</td>
<td>105</td>
</tr>
</tbody>
</table>
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
define v = QQ.valuation(2)
sage: v(1024)
10
```

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

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

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

```sage
define K.<a> = NumberField(x^2 + x + 1)
sage: w, ww = QQ.valuation(7).extensions(K)
sage: w(a + 3), ww(a + 3)
(1, 0)
sage: w(a + 5), ww(a + 5)
(0, 1)
```
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
```

Valuations can also be extended from smaller function fields:

```python
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]
```
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 $\mathbb{K}.\text{valuation}(x - 4)$ to the field $L$ above to outline how this works internally.

First, the valuation on $\mathbb{K}$ is induced by a valuation on $\mathbb{Q}[x]$. To construct this valuation, we start from the trivial valuation on $\mathbb{Q}$ and consider its induced Gauss valuation on $\mathbb{Q}[x]$, 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 $L$:

```
sage: R.<y> = K[]
sage: w = GaussValuation(R, v)
sage: w = w.augmentation(y - 2, 1); w
[ Gauss valuation induced by (x - 4)-adic valuation, v(y - 2) = 1 ]
sage: L.<y> = K.extension(y^2 - x)
sage: ww = w.extension(L); ww
[ (x - 4)-adic valuation, v(y - 2) = 1 ]-adic valuation
```
2.2 Limit valuations

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

```sage
sage: w_w(y^2 - x)
+Infinity
sage: y = R.gen()
sage: w(y^2 - x)
1
```

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

```sage
sage: w_w._base_valuation
[ Gauss valuation induced by \((x - 4)\)-adic valuation, \( v(y - 2) = 1 \), ... ]
```

The terms of this infinite sequence are computed on demand:

```sage
sage: w_w._base_valuation._approximation
[ Gauss valuation induced by \((x - 4)\)-adic valuation, \( v(y - 2) = 1 \) ]
sage: w_w(y - 1/4*x - 1)
2
sage: w_w._base_valuation._approximation
[ Gauss valuation induced by \((x - 4)\)-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
```
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: v = QQ.valuation(2)
sage: R.<x> = QQ[]
sage: f = x^5 + 3*x^4 + 5*x^3 + 8*x^2 + 6*x + 12
sage: v.mac_lane_approximants(f) # random output (order may vary)
[[ Gauss valuation induced by 2-adic valuation, v(x^2 + x + 1) = 3 ],
 [ Gauss valuation induced by 2-adic valuation, v(x) = 1/2 ],
 [ Gauss valuation induced by 2-adic valuation, v(x) = 1 ]]
```

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: v.mac_lane_approximants(f, required_precision=10) # random output
[[ Gauss valuation induced by 2-adic valuation, v(x^2 + 193*x + 13/21) = 10 ],
 [ Gauss valuation induced by 2-adic valuation, v(x + 86) = 10 ],
 [ Gauss valuation induced by 2-adic valuation, v(x) = 1/2, v(x^2 + 36/11*x + 2/17) = 11 ]]
```
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:**

```sage
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
class sage.rings.valuation.value_group.DiscreteValuationCodomain
    Bases: UniqueRepresentation, Parent

    The codomain of discrete valuations, the rational numbers extended by $\pm\infty$.

    EXAMPLES:

    ```sage```
    >>> 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 $\text{generator}$.

    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 $\mathbb{Z}$-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 $\mathbb{Q}$ appeared to be the better approach.

    **EXAMPLES:**
```
sage: 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

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

EXAMPLES:

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

gen()
Return a generator of this group.

EXAMPLES:

```python
sage: from sage.rings.valuation.value_group import DiscreteValueGroup
sage: DiscreteValueGroup(-3/8).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(0))
1
sage: DiscreteValueGroup(0).index(DiscreteValueGroup(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
```

**numerator()**

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
```

**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
```

**gens()**

Return the generators of this semigroup.

**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
```

**is_group()**

Return whether this semigroup is a group.

**EXAMPLES:**

```python
sage: from sage.rings.valuation.value_group import DiscreteValueSemigroup
sage: D1 = DiscreteValueSemigroup(0).is_group()
True
```
Discrete valuations can be created on a variety of rings:

```sage
sage: ZZ.valuation(2)
2-adic valuation
sage: GaussianIntegers().valuation(3)
3-adic valuation
sage: QQ.valuation(5)
5-adic valuation
sage: Zp(7).valuation()
7-adic valuation
```

### 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:

```sage
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
```

Invoking this method also changes the category of this semigroup if it is a group:

```sage
sage: D in AdditiveMagmas().AdditiveAssociative().AdditiveUnital().˓
    AdditiveInverse()
True
```

is_trivial()

Return whether this is the trivial additive abelian semigroup.

EXAMPLES:

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

some_elements()

Return some typical elements in this semigroup.

EXAMPLES:

```sage
sage: from sage.rings.valuation.value_group import DiscreteValueSemigroup
sage: list(DiscreteValueSemigroup([-3/8, 1/2]).some_elements())
[0, -3/8, 1/2, ...]
```
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

sage: R.<x> = QQ[]
sage: v = QQ.valuation(2)
# needs sage.rings.padics
sage: w = GaussValuation(R, v)
# needs sage.rings.padics
sage: w.augmentation(x, 3)
# needs sage.rings.padics
[ 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:

sage: w.augmentation(x, infinity)
# needs sage.rings.padics
[ Gauss valuation induced by 2-adic valuation, v(x) = +Infinity ]

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

is_equivalent(f, g)
Return whether f and g are equivalent.

EXAMPLES:
sage: # needs sage.rings.padics
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

class sage.rings.valuation.valuation.DiscreteValuation(parent)
Bases: DiscretePseudoValuation

5.2. Discrete valuations
Abstract base class for discrete valuations.

EXAMPLES:

```python
sage: v = QQ.valuation(2)  # needs sage.rings.padics
sage: R.<x> = QQ[]
```

```python
sage: v = GaussValuation(R, v)  # needs sage.rings.padics
```

```python
sage: w = v.augmentation(x, 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)
```

```python
sage: 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 or approximates 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)  # needs sage.rings.padics
```

```python
sage: R.<x> = QQ[]  # needs sage.rings.padics
```

```python
sage: G = x^2 + 1  # needs sage.rings.padics
```

We can select an approximant by approximating it:

```python
sage: w = GaussValuation(R, v).augmentation(x + 1, 1/2)  # needs sage.rings.padics
```

```python
sage: v.mac_lane_approximant(G, w)  # needs 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)  # needs sage.rings.padics
```

```python
sage: v.mac_lane_approximant(G, w)  # needs sage.rings.padics
```

(continues on next page)
needs sage.rings.padics
[ Gauss valuation induced by 2-adic valuation, \( v(x + 1) = \frac{1}{2} \) ]

Or it can be very specific:

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

But it must be an approximation of an approximant:

```python
sage: w = GaussValuation(R, v).augmentation(x, 1/2)  #
needs sage.rings.padics
sage: v.mac_lane_approximant(G, w)  #
eeds sage.rings.padics
Traceback (most recent call last):
...
ValueError: The valuation
[ Gauss valuation induced by 2-adic valuation, \( v(x) = \frac{1}{2} \) ] 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:

```python
sage: G = x^2 - 1  
needs sage.rings.padics
sage: w = GaussValuation(R, v)  
needs sage.rings.padics
sage: v.mac_lane_approximant(G, w)  
needs 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 \( x^2 - 1 \)
```

```python
sage: w = GaussValuation(R, v).augmentation(x + 1, 1)  
needs sage.rings.padics
sage: v.mac_lane_approximant(G, w)  
needs 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 \( x^2 - 1 \)
```

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


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

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

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

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
- \( \text{assume_squarefree} \) – a boolean (default: False), whether to assume that \( G \) is squarefree. If True, the squarefreeness of \( G \) is not verified though it is necessary when \( \text{require_final_EF} \) is set for the algorithm to terminate.
- \( \text{require_final_EF} \) – 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.
- \( \text{required_precision} \) – a number or infinity (default: -1); whether to require the last key polynomial of the returned valuations to have at least that valuation.
- \( \text{require_incomparability} \) – 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.)
- \( \text{required_degree} \) – 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.
- \( \text{algorithm} \) – one of "serial" or "parallel" (default: "serial"); whether or not to parallelize the algorithm

EXAMPLES:

```sage
sage: # needs sage.rings.padics
sage: v = QQ.valuation(2)
sage: R.<x> = QQ[]
sage: v.mac_lane_approximants(x^2 + 1)
[[ Gauss valuation induced by 2-adic valuation, v(x + 1) = 1/2 ]]
sage: v.mac_lane_approximants(x^2 + 1, required_precision=infinity)
[[ 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 ]]```
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 \( x - 1 \)):

\[
\text{sage: v.mac_lane_approximants}(x^2 - 1) \quad #\text{needs sage.rings.padics}
\]

\[
[[ \text{Gauss valuation induced by 2-adic valuation, } \nu(x + 1) = +\text{Infinity },] \\
[ \text{Gauss valuation induced by 2-adic valuation, } \nu(x + 1) = 1 ]] \\
\]

However, it needs to be squarefree:

\[
\text{sage: v.mac_lane_approximants}(x^2) \quad #\text{needs sage.rings.padics}
\]

Traceback (most recent call last):
...
ValueError: G must be squarefree

\[\text{montes_factorization}(G, \text{assume_squarefree}=\text{False}, \text{required_precision}={\text{None}})\]

Factor \( G \) over the completion of the domain of this valuation.

INPUT:

- \( G \) – a monic polynomial over the domain of this valuation
- \( \text{assume_squarefree} \) – a boolean (default: \text{False}), whether to assume \( G \) to be squarefree
- \( \text{required_precision} \) – 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 \( \text{required_precision} \).

ALGORITHM:

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

EXAMPLES:

\[
\text{sage: } \# \text{needs sage.rings.padics} \\
\text{sage: } k = \text{Qp}(5,4) \\
\text{sage: } v = k.valuation() \\
\text{sage: } R.<x> = k[] \\
\text{sage: } G = x^2 + 1 \\
\text{sage: } v.montes_factorization(G) \\
((1 + O(5^4)) \times x + 2 + 5 + 2 \times 5^2 + 5^3 + O(5^4)) \\
* ((1 + O(5^4)) \times x + 3 + 3 \times 5 + 2 \times 5^2 + 3 \times 5^3 + O(5^4)) \\
\]

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

\[
\text{sage: } R.<x> = \text{QQ}[x] \\
\text{sage: } v = \text{QQ}.valuation(2) \\
\quad \#\text{needs sage.rings.padics} \\
\text{sage: } v.montes_factorization(x^6 - 1) \\
\quad \#\text{needs sage.rings.padics} \\
(x - 1) \times (x + 1) \times (x^2 - x + 1) \times (x^2 + x + 1) \\
\text{sage: } v.montes_factorization(x^7 - 1) \quad \# \text{not tested} \\
\]

(continues on next page)
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:

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

```python
sage: # needs sage.rings.padics
sage: v = QQ.valuation(2)
sage: R.<x> = QQ[

is_discrete_valuation()

sage: v = GaussValuation(R, v)

sage: v.is_discrete_valuation()
True

sage: w = v.augmentation(x, infinity)

sage: w.is_discrete_valuation()
False
```

class sage.rings.valuation.valuation.MacLaneApproximantNode(valuation, parent, ef, principal_part_bound, coefficients, valuations)
Bases: object

A node in the tree computed by DiscreteValuation.mac_lane_approximants().

Leaves in the computation of the tree of approximants mac_lane_approximants(). Each vertex consists of a tuple \((v, ef, p, coeffs, vals)\) where \(v\) is an approximant, i.e., a valuation, \(ef\) is a boolean, \(p\) is the parent of this vertex, and \(coeffs\) and \(vals\) are cached values. (Only \(v\) and \(ef\) are relevant, everything else are caches/debug info.) The boolean \(ef\) denotes whether \(v\) already has the final ramification index \(E\) and residue degree \(F\) of this approximant. An edge \(V - P\) represents the relation \(P \cdot v \leq V \cdot v\) (pointwise on the polynomial ring \(K[x]\)) between the valuations.
class sage.rings.valuation.valuation.NegativeInfiniteDiscretePseudoValuation(parent)

Bases: InfiniteDiscretePseudoValuation

Abstract base class for pseudo-valuations which attain the value ∞ and −∞, i.e., whose domain contains an element of valuation ∞ and its inverse.

EXAMPLES:

```
sage: R.<x> = QQ[]
sage: v = GaussValuation(R, valuations.TrivialValuation(QQ)).augmentation(x, → infinity)
sage: K.<x> = FunctionField(QQ)
sage: w = K.valuation(v)
```

is_negative_pseudo_valuation()

Return whether this valuation attains the value −∞.

EXAMPLES:

```
sage: R.<x> = QQ[]
sage: u = GaussValuation(R, valuations.TrivialValuation(QQ))
sage: v = u.augmentation(x, infinity)
sage: v.is_negative_pseudo_valuation()  # False
sage: K.<x> = FunctionField(QQ)
sage: w = K.valuation(v)
sage: w.is_negative_pseudo_valuation()  # True
```

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 ∞.

AUTHORS:

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

EXAMPLES:

```
sage: QQ.valuation(2).parent()
Discrete pseudo-valuations on Rational Field
```

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):
```

(continues on next page)
AssertionError: False is not true

Instead, the valuations need to be created through the \_\_make\_element\_class\_\_ of the containing space:

\begin{verbatim}
sage: from sage.rings.valuation.trivial_valuation import TrivialDiscretePseudoValuation
sage: v = H.__make_element_class__(TrivialDiscretePseudoValuation)(H)
sage: v._test_category()
\end{verbatim}

The factories such as TrivialPseudoValuation provide the right inheritance structure:

\begin{verbatim}
sage: v = valuations.TrivialPseudoValuation(QQ)
sage: v._test_category()
\end{verbatim}

\begin{verbatim}
class sage.rings.valuation.valuation_space.DiscretePseudoValuationSpace(domain)
    Bases: UniqueRepresentation, Homset

    The space of discrete pseudo-valuations on domain.

    EXAMPLES:

    \begin{verbatim}
sage: from sage.rings.valuation.valuation_space import DiscretePseudoValuationSpace
sage: H = DiscretePseudoValuationSpace(QQ)
sage: QQ.valuation(2) in H
    True
    \end{verbatim}
\end{verbatim}

\textbf{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 in operator works which can essentially be achieved by overriding \_\_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 \_\_element\_constructor\_\_ to self, then one cannot register that conversion as a coercion. Consequently, the operators <= and => cannot be made to work between discrete valuations and discrete pseudo-valuations on the same domain (because the implementation only calls \_richcmp if both operands have the same parent.) Of course, we could override \_\_ge\_\_ and \_\_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.

\textbf{Todo:} The comparison problem might be fixed by github issue #22029 or similar.

\begin{verbatim}
class ElementMethods
    Bases: object

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

    EXAMPLES:

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

    \begin{verbatim}
sage: from sage.rings.valuation.valuation_space import DiscretePseudoValuationSpace
    \end{verbatim}
\end{verbatim}
Discrete Valuations and Discrete Pseudo-Valuations, Release 10.1

(continued from previous page)

sage: H = DiscretePseudoValuationSpace(QQ)
sage: QQ.valuation(2).is_discrete_pseudo_valuation()  # indirect doctest
True

The methods will be provided even if the concrete type is not created with \texttt{\_\_make\_element\_class\_}:

sage: from sage.rings.valuation.valuation import DiscretePseudoValuation
sage: m = DiscretePseudoValuation(H)
sage: m.parent() is H
True
sage: m.is_discrete_pseudo_valuation()
True

However, the category framework advises you to use inheritance:

sage: m\._test\_category()
Traceback (most recent call last):
  ...
AssertionError: False is not true

Using \texttt{\_\_make\_element\_class\_}, makes your concrete valuation inherit from this class:

sage: m = H\._\_make\_element\_class\_\_(DiscretePseudoValuation)(H)
sage: m\._test\_category()

\texttt{\_\_change\_domain\_\_(ring)}

Return this valuation over \texttt{ring}.

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

\textbf{EXAMPLES:}

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

\texttt{\_\_element\_with\_valuation\_\_(s)}

Return an element in the domain of this valuation with valuation \texttt{s}.

\textbf{EXAMPLES:}

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

\texttt{\_\_extension\_\_(ring)}

Return the unique extension of this valuation to \texttt{ring}.

\textbf{EXAMPLES:}

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

5.3. \textit{Spaces of valuations}
**extensions**(*ring*)

Return the extensions of this valuation to *ring*.

**EXAMPLES:**

```
sage: v = ZZ.valuation(2)
sage: 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
```

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
```

Of course this always works over fields:

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

**is_discrete_pseudovaluation**()

Return whether this valuation is a discrete pseudo-valuation.

**EXAMPLES:**

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

**is_discretevaluation**()

Return whether this valuation is a discrete valuation, i.e., whether it is a discrete pseudo valuation that only sends zero to ∞.

**EXAMPLES:**

```
sage: QQ.valuation(2).is_discretevaluation()
True
```
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: QQ.valuation(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: QQ.valuation(7).is_trivial()
False
```

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

EXAMPLES:
```
sage: v = QQ.valuation(2)
sage: 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: v = ZZ.valuation(2)
sage: v.lower_bound(2^10)
10
```

reduce($x$)
Return the image of $x$ in the 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:

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

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

```
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 (using ...)
```

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

EXAMPLES:

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

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

INPUT:
- \( \text{scalar} \) – a non-negative rational number or infinity

EXAMPLES:

```
sage: v = ZZ.valuation(3)
sage: w = v.scale(3)
sage: w(3)
3
```
Scaling can also be done through multiplication with a scalar:

```
sage: w/3 == v
True
```

Multiplication by zero produces the trivial discrete valuation:

```
sage: w = 0*v
sage: w(3)
0
sage: w(0)
+Infinity
```

Multiplication by infinity produces the trivial discrete pseudo-valuation:

```
sage: w = infinity*v
sage: w(3)
+Infinity
sage: w(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:**

```
sage: v2 = QQ.valuation(2)
sage: v3 = QQ.valuation(3)
sage: v5 = QQ.valuation(5)
sage: v2.separating_element([v3,v5])
4/15
```

**shift(x, s)**

Shift `x` in its expansion with respect to `uniformizer()` by `s` “digits”.

For non-negative `s`, this just returns `x` multiplied by a power of the uniformizer `π`.

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

**EXAMPLES:**

```
sage: v = ZZ.valuation(2)
sage: v.shift(1, 10)
1024
sage: v.shift(11, -1)
5
```

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

```
sage: R.<x> = ZZ[]
sage: v = ZZ.valuation(2)
sage: w = GaussValuation(R, v)
sage: w.shift(x, 1)
```

(continues on next page)
Discrete Valuations and Discrete Pseudo-Valuations, Release 10.1

2*x
sage: w.shift(2*x, -1)
x
sage: w.shift(x + 2*x^2, -1)
x^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: 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
```

**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
```
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
```

A pseudo-valuation that is $\infty$ 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.
```

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:

```python
sage: G = QQ.valuation(2).value_semigroup(); G
Additive Abelian Semigroup generated by -1, 1
```

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

```python
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:

```python
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
```

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
```

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
```

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
```

reduce(x)

Reduce $x$ modulo the positive elements of this valuation.

EXAMPLES:

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

Return the value group of this valuation.

EXAMPLES:

A trivial discrete pseudo-valuation has no value group:

```
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.
```

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

Bases: DiscretePseudoValuation

Base class for code shared by trivial valuations.

EXAMPLES:

```
sage: 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
```

is_trivial()

Return whether this valuation is trivial.

EXAMPLES:

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

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

Bases: TrivialDiscretePseudoValuation_base, DiscreteValuation

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

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

extensions(ring)
Return the unique extension of this valuation to ring.

EXAMPLES:

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

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

EXAMPLES:

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

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

EXAMPLES:

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

residue_ring()
Return the residue ring of this valuation.

EXAMPLES:

sage: 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

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

Bases: UniqueFactory
Create a trivial valuation on domain.

EXAMPLES:
create_key(domain)

Create a key that identifies this valuation.

EXAMPLES:

```python
c sage: valuations.TrivialValuation(QQ) is valuations.TrivialValuation(QQ) # indirect doctest
True
```

create_object(version, key, **extra_args)

Create a trivial valuation from key.

EXAMPLES:

```python
c sage: valuations.TrivialValuation(QQ) # indirect doctest
Trivial valuation on Rational Field
```

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:


EXAMPLES:

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

```python
c sage: R.<x> = QQ[]
c sage: v0 = QQ.valuation(2)
c 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:

```python
c sage: v = v.augmentation(x, 1/4); v
[ Gauss valuation induced by 2-adic valuation, v(x) = 1/4 ]
c sage: v = v.augmentation(x^4 + 2^x^3 + 2^x^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 ]
c sage: S.<T> = R[]
c sage: w = GaussValuation(S, v); w
Gauss valuation induced by [ 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 ]
```

(continues on next page)
class sage.rings.valuation.gauss_valuation.GaussValuationFactory

Bases: UniqueFactory

Create a Gauss valuation on domain.

INPUT:

• domain – a univariate polynomial ring
• v – a valuation on the base ring of domain, the underlying valuation on the constants of the polynomial ring (if unspecified take the natural valuation on the valued ring domain.)

EXAMPLES:

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

```sage
c = QQ.valuation(2)
c
R.<x> = QQ[]
c = GaussValuation(R, c)
c(2)
1
0
c(x)
0
0
c(x + 2)
```

create_key(domain, 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:

```sage
S = Zp(3,5)
S.<x> = S[]
S0 = S.valuation()
S = GaussValuation(S, S0); S
Gauss valuation induced by 3-adic valuation
S = S.<x> = QQ[]
S = GaussValuation(S, QQ.valuation(5)); S
Gauss valuation induced by 5-adic valuation
```
E()

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

EXAMPLES:

```sage
sage: R.<u> = Qq(4,5)
sage: S.<x> = R[]
sage: v = GaussValuation(S)
sage: v.E()
1
```

F()

Return the degree of the residue field extension of this valuation over the Gauss valuation, i.e., 1.

EXAMPLES:

```sage
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:

```sage
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(ring)

Return this valuation as a valuation over ring.

EXAMPLES:

```sage
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
```

element_with_valuation(s)

Return a polynomial of minimal degree with valuation s.

EXAMPLES:

```sage
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.

5.5. Gauss valuations on polynomial rings 33
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: 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]
```

`is_gauss_valuation()`

Return whether this valuation is a Gauss valuation.

EXAMPLES:

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

`is_trivial()`

Return whether this is a trivial valuation (sending everything but zero to zero.)

EXAMPLES:

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

`lift(F)`

Return a lift of \( F \).

INPUT:

- \( F \) – a polynomial over the `residue_ring()` of this valuation

OUTPUT:

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

EXAMPLES:
Discrete Valuations and Discrete Pseudo-Valuations, Release 10.1

```
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:

```
sage: R.<u> = QQ
sage: S.<x> = R[

sage: v = GaussValuation(S, QQ.valuation(2))

sage: y = v.residue_ring().gen()

sage: f = v.lift_to_key(y^2 + y + 1); f
x^2 + x + 1
```

lower_bound($f$)

Return an 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: R.<u> = Qq(4, 5)

sage: S.<x> = R[

sage: v = GaussValuation(S)

sage: v.lower_bound(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:

```
reduce($f$, check=True, degree_bound=None, coefficients=None, valuations=None)

Return the reduction of $f$ modulo this valuation.

INPUT:

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

OUTPUT:

A polynomial in the residue_ring() of this valuation.

EXAMPLES:

```
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
```

The reduction is only defined for integral elements:

```
sage: f = x^2/2
sage: v.reduce(f)
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:

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

**restriction** *(ring)*

Return the restriction of this valuation to ring.

EXAMPLES:

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

**scale** *(scalar)*

Return this valuation scaled by scalar.

EXAMPLES:

```sage
sage: R.<x> = QQ[]
sage: v = GaussValuation(R, QQ.valuation(2))
sage: 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
sage: R.<u> = Qq(4, 5)
sage: S.<x> = R[]
(sage: v = GaussValuation(S)
(sage: f = x^10/2 + 1
```
sage: v.simplify(f)
\((2^{-1} + O(2^4)) \times 10 + 1 + O(2^5)\)

**uniformizer()**

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

**EXAMPLES:**

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

```python
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
```

**valuations(f, coefficients=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
- \(coefficients\) – 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:**

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

**EXAMPLES:**

```python
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]
```
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
```

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
```

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:

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

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

```sage
f = x^2 + 2*x + 2
sage: 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:

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

Another example of a DevelopingValuation is an augmented valuation:

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

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

```sage
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))
```

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: R = Qp(2,5)
sage: S.<x> = R[]
sage: v = GaussValuation(S)
sage: f = x^2 + 2*x + 3
sage: 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: 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
```

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: R = Qp(2,5)
sage: S.<x> = R[]
sage: v = GaussValuation(S)
sage: f = x^2 + 2*x + 3
sage: v.newton_polygon(f)
Finite Newton polygon with 2 vertices: (0, 0), (2, 0)

sage: v = v.augmentation( x^2 + x + 1, 1)
sage: v.newton_polygon(f)
Finite Newton polygon with 2 vertices: (0, 0), (1, 1)

sage: v.newton_polygon( f * v.phi()^3 )
Finite Newton polygon with 2 vertices: (3, 3), (4, 4)
```

**phi()**

Return the polynomial \( \phi \), the key polynomial of this valuation.

**EXAMPLES:**

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

```
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:

```
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 [Rüt2014].

```python
class sage.rings.valuation.inductive_valuation.FinalInductiveValuation(parent, phi)
    Bases: InductiveValuation
    Abstract base class for an inductive valuation which cannot be augmented further.

class sage.rings.valuation.inductive_valuation.FiniteInductiveValuation(parent, phi)
    Bases: InductiveValuation, DiscreteValuation
    Abstract base class for iterated augmented valuations on top of a Gauss valuation which is a discrete valuation, i.e., the last key polynomial has finite valuation.

    EXAMPLES:
    sage: R.<x> = QQ[]
    sage: v = GaussValuation(R, valuations.TrivialValuation(QQ))
    sage: extensions(other)
    Return the extensions of this valuation to other.

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

\begin{Verbatim}
sage: R.<x> = QQ[]
sage: v = GaussValuation(R, QQ.valuation(5))
\end{Verbatim}

\textbf{E()}

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

\textbf{EXAMPLES:}

\begin{Verbatim}
sage: R.<u> = Qq(4,5)
sage: S.<x> = R[]
sage: v = GaussValuation(S)
sage: v.E()
1
\end{Verbatim}

\textbf{F()}

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

\textbf{EXAMPLES:}

\begin{Verbatim}
sage: R.<u> = Qq(4,5)
sage: S.<x> = R[]
sage: v = GaussValuation(S)
sage: v.F()
1
\end{Verbatim}

\textbf{augmentation_chain()}

Return a list with the chain of augmentations down to the underlying \textit{Gauss valuation}.

\textbf{EXAMPLES:}

\begin{Verbatim}
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]
\end{Verbatim}

\textbf{element_with_valuation(s)}

Return a polynomial of minimal degree with valuation \textit{s}.

\textbf{EXAMPLES:}

\begin{Verbatim}
sage: R.<x> = QQ[]
sage: v = GaussValuation(R, QQ.valuation(2))
sage: v.element_with_valuation(-2)
1/4
\end{Verbatim}

Depending on the base ring, an element of valuation \textit{s} might not exist:

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

5.7. Inductive valuations on polynomial rings
equivalence_reciprocal\((f,\text{coefficients}=\text{None}, \text{valuations}=\text{None}, \text{check}=\text{True})\)

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()
- coefficients – the coefficients of \(f\) in the phi()-adic expansion if known (default: None)
- valuations – the valuations of coefficients if known (default: None)
- 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 xgcd implementation there.

**EXAMPLES:**

```
sage: R = Zp(3,5)
sage: S.<x> = R[]
sage: v = GaussValuation(S)
sage: f = 3^2*x + 2
sage: h = v.equivalence_reciprocal(f); h
\(2 + O(3^5)\)
sage: v.is_equivalent(f*h, 1)
True
```

In an extended valuation over an extension field:

```
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 = 2^2*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:

```
sage: v = v.augmentation((x^2 + x + u)^2 + 2*(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 value_group()
Discrete Valuations and Discrete Pseudo-Valuations, Release 10.1

- **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
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
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...
```

**is_equivalence_unit**\((f, \text{valuations}=\text{None})\)

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
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
```

**is_gauss_valuation**\()

Return whether this valuation is a Gauss valuation over the domain.

**EXAMPLES:**

```sage
sage: R.<u> = Qq(4,5)

sage: S.<x> = R[

sage: v = GaussValuation(S)

sage: v.is_gauss_valuation()
True
```

5.7. Inductive valuations on polynomial rings
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:

```python
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)
```

mu()
Return the valuation of $\phi()$.

EXAMPLES:

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

class sage.rings.valuation.inductive_valuation.InfiniteInductiveValuation(
    parent, base_valuation)

Bases: FinalInductiveValuation, InfiniteDiscretePseudoValuation

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

EXAMPLES:

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

change_domain(ring)
Return this valuation over ring.

EXAMPLES:

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

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

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

Bases: FinalInductiveValuation, DiscreteValuation

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

EXAMPLES:
augmentation\((\phi, \mu, \text{check=}\text{True})\)

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 \texttt{is_key()} for properties of key polynomials.
- \(\mu\) – a rational number or infinity, the valuation of \(\phi\) in the extended valuation
- \texttt{check} – a boolean (default: \texttt{True}), whether or not to check the correctness of the parameters

**EXAMPLES:**

```python
sage: R.<u> = QQ(4,5)
sage: S.<x> = R[]
sage: v = GaussValuation(S)
sage: v = v.augmentation(x^2 + x + u, 1)
sage: 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^2 + (2*2 + O(2^6))*x^3 + (1 + O(2^5))*x^2 + (u + O(2^5))) = 3
```

See also:

\texttt{augmented_valuation}

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

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

**INPUT:**

- \(f\) – a non-zero polynomial in the domain of this valuation
- \texttt{assume_not_equivalence_unit} – whether or not to assume that \(f\) is not an equivalence unit (default: \texttt{False})
- \texttt{coefficients} – the coefficients of \(f\) in the \texttt{phi()}-adic expansion if known (default: \texttt{None})
- \texttt{valuations} – the valuations of coefficients if known (default: \texttt{None})
- \texttt{compute_unit} – whether or not to compute the unit part of the decomposition (default: \texttt{True})
- \texttt{degree_bound} – a bound on the degree of the \texttt{_equivalence_reduction()} of \(f\) (default: \texttt{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 \texttt{residue_field()}. Lifting all \(\alpha_i\) to key polynomials \(\phi_i\) gives \(Rf = \prod_i R_i f_i\) for suitable
equivalence units $R_i$ (see \texttt{lift_to_key()}). Taking $R'$ an \texttt{equivalence_reciprocal()} of $R$, we have $f$ equivalent to $(R' \prod_i R_i) \prod_i \phi_i$.

EXAMPLES:

```python
sage: R.<u> = Qq(4,10)
sage: S.<x> = R[]
sage: v = GaussValuation(S)
sage: v.equivalence_decomposition(S.zero())
Traceback (most recent call last):
  ...  
ValueError: equivalence decomposition of zero is not defined
sage: v.equivalence_decomposition(S.one())
1 + O(2^10)
sage: v.equivalence_decomposition(x^2+2)
((1 + O(2^10))^x*x)^2
sage: v.equivalence_decomposition(x^2+1)
((1 + O(2^10))^x + 1 + O(2^10))^2
```

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

```python
sage: w = v.augmentation(x, 1)
sage: F = w.equivalence_decomposition(x^2+1); F
(1 + O(2^10))*x^2 + 1 + O(2^10)
sage: F.unit()
(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:

```python
sage: f = x * (x^2 + 1)
sage: F = w.equivalence_decomposition(f); F
(1 + O(2^10))^x^2
sage: F.unit()
1 + O(2^10)
```

Examples over an iterated unramified extension:

```python
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 )
sage: len(F)
2
```

\texttt{is_equivalence_irreducible(f, coefficients=None, valuations=None)}

Return whether the polynomial $f$ is equivalence-irreducible, i.e., whether its \texttt{equivalence_decomposition()} is trivial.

ALGORITHM:
We use the same algorithm as in `equivalence_decomposition()` we just do not lift the result to key polynomials.

INPUT:

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

EXAMPLES:

```python
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
```

```python
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_equivalence_irreducible()`, and whether it `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:

```python
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')
```

```python
is_minimal(f, assume_equivalence_irreducible=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 \( fc \ is_equivalent() \) to \( h \).

ALGORITHM:

Based on Theorem 9.4 of [Mac1936II].

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

\textbf{lift\_to\_key}(F)

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

\textbf{INPUT:}

- \(F\) \textendash an irreducible non-constant monic polynomial in \texttt{residue\_ring()} of this valuation

\textbf{OUTPUT:}

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

More specifically, if \(F\) is not the generator of the residue ring, then multiplying \(f\) with the \texttt{equivalence\_reciprocal()} of the \texttt{equivalence\_unit()} of the valuation of \(f\), produces a unit which reduces to \(F\).

\textbf{EXAMPLES:}

\begin{verbatim}
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)
\end{verbatim}

\textbf{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 \texttt{equivalence\_unit}.

This performs the individual steps that are used in \texttt{mac\_lane\_approximants()}.

\textbf{INPUT:}

- \(G\) \textendash a squarefree monic non-constant integral polynomial \(G\) which is not an \texttt{equivalence\_unit}
- \texttt{principal\_part\_bound} \textendash an integer or \texttt{None} (default: \texttt{None}), a bound on the length of the principal part, i.e., the section of negative slope, of the Newton polygon of \(G\)
- \texttt{assume\_squarefree} \textendash whether or not to assume that \(G\) is squarefree (default: \texttt{False})
- \texttt{assume\_equivalence\_irreducible} \textendash whether or not to assume that \(G\) is equivalence irreducible (default: \texttt{False})
- \texttt{report\_degree\_bounds\_and\_caches} \textendash whether or not to include internal state with the returned value (used by \texttt{mac\_lane\_approximants()} to speed up sequential calls)
- \texttt{coefficients} \textendash the coefficients of \(G\) in the \texttt{phi()}-adic expansion if known (default: \texttt{None})
- \texttt{valuations} \textendash the valuations of \texttt{coefficients} if known (default: \texttt{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 `mac_lane_approximants()`:

```python
sage: R.<x> = QQ[]
sage: v = QQ.valuation(2)
sage: f = x^36 + 1160/81*x^31 + 9920/27*x^30 + 1040/81*x^26 + 52480/81*x^25 +
    ... 220160/81*x^24 - 5120/81*x^21 - 143360/81*x^20 - 573440/81*x^19 + 12451840/
    ... 81*x^18 - 266240/567*x^16 - 20316160/567*x^15 - 198737920/189*x^14 -
    ... 1129840640/81*x^13 - 1907359744/27*x^12 + 8192/81*x^11 + 655360/21*x^10 +
    ... 5242880/21*x^9 + 2118123520/567*x^8 + 1546024544/567*x^7 + 6509559808/81*x^6 +
    ... 16777216/567*x^2 - 268435456/567*x - 1073741824/567
sage: v.mac_lane_approximants(f)
[[ Gauss valuation induced by 2-adic valuation, v(x + 2056) = 23/2 ],
[ Gauss valuation induced by 2-adic valuation, v(x) = 11/9 ],
[ Gauss valuation induced by 2-adic valuation, v(x) = 2/5, v(x^5 + 4) = 7/2 ],
[ 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) = 3/5, v(x^5 + 8) = 5 ]]
```

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) = 3/5 ],
[ 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:

```python
sage: V1[1].mac_lane_step(f)
[[ 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 \( \beta V1[1] \). To ignore such trivial augmentations, we can set `allow_equivalent_key`:

```python
sage: V1[1].mac_lane_step(f, allow_equivalent_key=False)
[[ 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_representation(/)
Return a minimal representative for \( f \), i.e., a pair \( e, a \) such that \( f \) \text{ is\_equivalent() } to \( ea \), \( e \) is an \textit{equivalence unit}, and \( a \) \text{ is\_minimal() } and monic.

\textbf{INPUT:}
- \( f \) – a non-zero polynomial which is not an equivalence unit

\textbf{OUTPUT:}
A factorization which has \( e \) as its unit and \( a \) as its unique factor.

\textbf{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 \textit{effective\_degree()}). Let \( h \) be the \textit{equivalence\_reciprocal()} of \( e \) and take \( a \) given by the terms of minimal valuation in the expansion of \( ef \).

\textbf{EXAMPLES:}

\begin{verbatim}
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: 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
\end{verbatim}

5.8 Augmented valuations on polynomial rings

Implements augmentations of (inductive) valuations.

\textbf{AUTHORS:}

\textbf{EXAMPLES:}
Starting from a \textit{Gauss valuation}, we can create augmented valuations on polynomial rings:

\begin{verbatim}
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
\end{verbatim}
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:

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

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.

```python
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 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:

```
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

E()

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

EXAMPLES:

```
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: R.<u> = Qq(4, 5)
sage: S.<x> = R[]
sage: v = GaussValuation(S)
sage: w = v.augmentation(x^2 + x + u, 1)
sage: w.F()
2
sage: w = v.augmentation(x, 1/2)
sage: w.F()
1
```

augmentation_chain()

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

EXAMPLES:

```
sage: R.<x> = QQ[]
sage: v = GaussValuation(R, QQ.valuation(2))
sage: w = v.augmentation(x, 1)
sage: w.augmentation_chain()
(continues on next page)
```
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:

```
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:

```
sage: R.<x> = GaussianIntegers()

sage: v = GaussValuation(R, GaussianIntegers().valuation(2))
sage: v = v.augmentation(x, 1)
sage: v.change_domain(QQ['x'])
[ Gauss valuation induced by z-adic valuation, v(x) = 1 ]
```

**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: 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.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):
...
```

(continues on next page)
equivalence_unit\( (s, \text{reciprocal}=False) \)

Return an equivalence unit of minimal degree and valuation \( s \).

**INPUT:**

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

**OUTPUT:**

A polynomial in the domain of this valuation which \( \text{is_equivalence_unit()} \) for this valuation.

**EXAMPLES:**

```
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)
```

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:

```
sage: w = v.augmentation(x, 1/2)
sage: w.equivalence_unit(3/2)
Traceback (most recent call last):
...
ValueError: 3/2 is not in the value semigroup of 2-adic valuation
sage: w.equivalence_unit(1)
2 + O(2^6)
```

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

```
sage: w = v.augmentation(x, 3/4)
sage: ww = w.augmentation(x^4 + 8, 5)

sage: ww.equivalence_unit(1/2)
(2^{-1} + O(2^4))*x^2
```

extensions\( (\text{ring}) \)

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

**EXAMPLES:**

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

\textbf{is\_gauss\_valuation()}

Return whether this valuation is a Gauss valuation.

\textbf{EXAMPLES:}

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

\textbf{is\_negative\_pseudo\_valuation()}

Return whether this valuation attains $-\infty$.

\textbf{EXAMPLES:}

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

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

\textbf{is\_trivial()}

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

\textbf{EXAMPLES:}

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

\textbf{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.

\textbf{EXAMPLES:}

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 (continues on next page))

\section{5.8. Augmented valuations on polynomial rings}
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: 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
```

restriction(ring)

Return the restriction of this valuation to ring.

EXAMPLES:

```
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)
sage: w.restriction(QQ['x'])
[ Gauss valuation induced by 2-adic valuation, v(x^2 + x + 1) = 1 ]
```

scale(scalar)

Return this valuation scaled by scalar.

EXAMPLES:

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

sage: w.uniformizer()
2
```

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

Bases: AugmentedValuation_base, FinalInductiveValuation

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

EXAMPLES:

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

sage: w.lift(1/2)
1/2
```

```python
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:

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

sage: w.lift(w.residue_ring().gen())
x
```

A case with non-trivial base valuation:

```python
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())
(1 + O(2^10))*x
```
```
**reduce**\((f,\text{check}=\text{True, degree\_bound}={\text{None}},\text{coefficients}={\text{None, valuations}=\text{None}})\)

Reduce \(f\) module 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 \text{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 \text{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)
sage: w.reduce(x^2 + x + 1)
1

sage: w = v.augmentation(x^2 + x + 1, infinity)
sage: w.reduce(x)
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:**

```python
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()
Number Field in u1 with defining polynomial x^2 + x + 1
```

An example with a non-trivial base valuation:

```python
sage: v = GaussValuation(R, QQ.valuation(2))
sage: w = v.augmentation(x^2 + x + 1, infinity)
sage: w.residue\_ring()
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
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
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: R.<u> = Qq(4, 5)
sage: S.<x> = R[]
sage: v = GaussValuation(S)
sage: w = v.augmentation(x^2 + x + u, 1/2)
```

```python
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 `coefficients()` of the \( \phi() \)-adic expansion of \( f \) (which often leads to coefficient bloat.) So unless \( \phi() \) is trivial, we fall back to valuation which this valuation augments since it is guaranteed to be smaller everywhere.

EXAMPLES:

```python
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)
```

```
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 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 in the $\phi()$-adic development 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 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:

```sage
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 github issue #25607 has been resolved, i.e., the coefficients in the following example are small:

```sage
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 ]
```

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

EXAMPLES:

```sage
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.upper_bound(x^2 + x + u)
1/2
```

valuations($f$, coefficients=None, call_error=False)

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

62 Chapter 5. More Details
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) \([v(f_0), v(f_1), \ldots]\)

EXAMPLES:

```sage
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: list(w.valuations( x^2 + 1 ))
[0, 1/2]
sage: ww = w.augmentation((x^2 + x + u)^2 + 2, 5/3)
sage: list(ww.valuations( ((x^2 + x + u)^2 + 2)^3 ))
[+Infinity, +Infinity, +Infinity, 5]
```

`value_group()`

Return the value group of this valuation.

EXAMPLES:

```sage
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.value_group()
Additive Abelian Group generated by 1/2
sage: ww = w.augmentation((x^2 + x + u)^2 + 2, 5/3)
sage: ww.value_group()
Additive Abelian Group generated by 1/6
```

`value_semigroup()`

Return the value semigroup of this valuation.

EXAMPLES:

```sage
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()
(continues on next page)
```
Additive Abelian Semigroup generated by 1/2

```python
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
```

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:**

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

**lower_bound**

```
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:**

```python
sage: R.<u> = Qq(4, 5)
sage: S.<x> = R[]
sage: v = GaussValuation(S)
sage: w = v.augmentation(x^2 + x + u, infinity)
sage: w.lower_bound(x^2 + x + u)
+Infinity
```

**simplify**

```
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:**

```python
sage: R.<u> = Qq(4, 5)
sage: S.<x> = R[]
sage: v = GaussValuation(S)
sage: w = v.augmentation(x^2 + x + u, infinity)
sage: w.simplify(x^10/2 + 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: R.<u> = QQ(4, 5)
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
```

valuations($f$, 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) $[v(f_0), v(f_1 \phi), ...]$

EXAMPLES:

```
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]
```

value_group()

Return the value group of this valuation.

EXAMPLES:

```
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
```

value_semigroup()

Return the value semigroup of this valuation.

EXAMPLES:

```
sage: R.<u> = QQ(4, 5)
sage: S.<x> = R[]
sage: v = GaussValuation(S)
sage: w = v.augmentation(x, infinity)
sage: w.value_semigroup()
Additive Abelian Group generated by 1
```
Discrete Valuations and Discrete Pseudo-Valuations, Release 10.1

```python
sage: R.<u> = Zq(4, 5)
sage: S.<x> = R[]
sage: v = GaussValuation(S)
sage: w = v.augmentation(x, infinity)
sage: w.value_semigroup()
Additive Abelian Semigroup generated by 1
```

```python
class sage.ringsvaluation.augmentedvaluation.NonFinalAugmentedValuation(parent, v, phi, mu)
Bases: AugmentedValuation_base, NonFinalInductiveValuation
An augmented valuation which can be augmented further.

EXAMPLES:

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

```python
lift(F, report_coefficients=False)
Return a polynomial which reduces to F.

INPUT:

- F – an element of the `residue_ring`
- report_coefficients – whether to return the coefficients of the `phi`-adic expansion or the actual polynomial (default: False, i.e., return the polynomial)

OUTPUT:

A polynomial in the domain of the valuation with reduction F, monic if F is monic.

ALGORITHM:
Since this is the inverse of `reduce`, we only have to go backwards through the algorithm described there.

EXAMPLES:

```python
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: 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
```
```
```
```
(continued from previous page)

```python
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
```

A more complicated example:

```python
sage: v = GaussValuation(S)
sage: w = v.augmentation(x^2 + x + u, 1/2)
sage: ww = w.augmentation((x^2 + x + u)^2 + 2*x*(x^2 + x + u) + 4*x, 3)
sage: u = ww.residue_ring().base().gen()

sage: F = ww.residue_ring()(u); F
u2
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)

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 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:**

We follow the algorithm described in Theorem 13.1 [Mac1936I] which, after a lift() of F, essentially shifts the valuations of all terms in the 𝜑-adic expansion up and then kills the leading coefficient.

**EXAMPLES:**

```python
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(2^10))*x + (u + 1) + u*2 + O(2^10)
```

(continues on next page)
A more complicated example:

```python
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: u = ww.residue_ring().base().gen()
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
```

In the `reduce` function, we have:

```python
reduce(f, check=True, degree_bound=None, coefficients=None, valuations=None)
```

Reduce `f` modulo this valuation.

**INPUT:**

- `f` – an element in the domain of this valuation
- `check` – whether or not to check whether `f` has non-negative valuation (default: `True`)
- `degree_bound` – an a-priori known bound on the degree of the result which can speed up the computation (default: not set)
- `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.
- `valuations` – the valuations of `coefficients` or `None` (default: `None`)

**OUTPUT:**

an element of the `residue_ring()` of this valuation, the reduction modulo the ideal of elements of positive valuation

**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 `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 `equivalence_unit()` \( Q \) with the same valuation as \( \phi^\tau \). Let \( Q' \) be its `equivalence_reciprocal()`. Now, rewrite each term \( f_i \phi^{i\tau} = (f_i Q') (\phi^\tau Q'^{-1})^i \); it turns out that the second factor in this expression is a lift of the generator of the `residue_field()`. The reduction of the first factor can be computed recursively.

**EXAMPLES:**

```python
sage: R.<u> = Qq(4, 10)
sage: S.<x> = R[]
sage: v = GaussValuation(S)
sage: v.reduce(x)
x
sage: v.reduce(S(u))
```

(continues on next page)
\begin{verbatim}
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 of w over v
u1
sage: f = (x^2 + x + u)^2 / 2
sage: w.reduce(f)
x
sage: w.reduce(f + x + 1)
x + u1 + 1
sage: ww = w.augmentation((x^2 + x + u)^2 + 2, 5/3)
sage: g = ((x^2 + x + u)^2 + 2)^3 / 2^5
sage: ww.reduce(g)
x
sage: ww.reduce(f)
1
sage: ww.is_equivalent(f, 1)
True
sage: ww.reduce(f * g)
x
sage: ww.reduce(f + g)
x + 1
\end{verbatim}

\section*{residue_ring()}
Return the residue ring of this valuation, i.e., the elements of non-negative valuation modulo the elements of positive valuation.

\textbf{EXAMPLES:}

\begin{verbatim}
sage: R.<x> = QQ[]
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
\end{verbatim}

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

\begin{verbatim}
sage: w = v.augmentation(x, 1)
sage: w.residue_ring()
Univariate Polynomial Ring in x over Finite Field of size 2 (using ...)
\end{verbatim}

\section*{class sage.rings.valuation.augmented_valuation.NonFinalFiniteAugmentedValuation}
Bases: \texttt{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
sage: R.<x> = QQ[]

sage: v = GaussValuation(R, QQ.valuation(2))

sage: w = v.augmentation(x, 1)
```

### 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:

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

sage: R.<y> = K[]

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:

```sage
sage: K = QQ

sage: R.<t> = K[]

sage: L.<t> = 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]
```

*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:
Discrete Valuations and Discrete Pseudo-Valuations, Release 10.1

\texttt{sage: K.<x> = FunctionField(QQ)}
\texttt{sage: R.<y> = K[\]}  
\texttt{sage: L.<y> = K.extension(y^2 - x)}
\texttt{sage: v = K.valuation(1/x)}
\texttt{sage: w = v.extension(L); w}  
\texttt{Valuation at the infinite place}
\texttt{sage: w._base_valuation._base_valuation._improve_approximation()}  
\texttt{sage: w._base_valuation._base_valuation._approximation}  
\text{[ Gauss valuation induced by Valuation at the infinite place, v(y) = 1/2, v(y^2 - 1/x) \rightarrow +\text{Infinity} ]}

REFERENCES:

Limits of inductive valuations are discussed in [Mac1936I] and [Mac1936II]. An overview can also be found in Section 4.6 of [Rüt2014].

\texttt{class sage.rings.valuation.limit_valuation.LimitValuationFactory}
\texttt{Bases: UniqueFactory}

Return a limit valuation which sends the polynomial \( G \) to infinity and is greater than or equal than \( \text{base_valuation} \).

INPUT:

- \( \text{base_valuation} \) – a discrete (pseudo-)valuation on a polynomial ring which is a discrete valuation on the coefficient ring which can be uniquely augmented (possibly only in the limit) to a pseudo-valuation that sends \( G \) to infinity.
- \( G \) – a squarefree polynomial in the domain of \( \text{base_valuation} \).

EXAMPLES:

\texttt{sage: R.<x> = QQ[]}  
\texttt{sage: v = GaussValuation(R, QQ.valuation(2))}  
\texttt{sage: w = valuations.LimitValuation(v, x)}  
\texttt{sage: w(x)}  
\texttt{+Infinity}

\texttt{create_key}(\text{base\_valuation}, \text{G})

Create a key from the parameters of this valuation.

EXAMPLES:

Note that this does not normalize \( \text{base\_valuation} \) in any way. It is easily possible to create the same limit in two different ways:

\texttt{sage: R.<x> = QQ[]}  
\texttt{sage: v = GaussValuation(R, QQ.valuation(2))}  
\texttt{sage: w = valuations.LimitValuation(v, x)}  
\texttt{sage: v = v.augmentation(x, infinity)}  
\texttt{sage: u = valuations.LimitValuation(v, x)}  
\texttt{sage: u == w}  
\texttt{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.

5.9. Valuations which are defined as limits of valuations.
create_object\((\text{version, key})\)

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\((\text{parent, approximation})\)

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: R.<x> = QQ[

```

```
sage: L.<y>=K.extension(y^2 - x)
```

```
sage: v = K.valuation(0)
sage: w = v.extension(L)
sage: 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\((f, \text{check=True})\)

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
- \(\text{check}\) – whether or not to check that \(f\) has non-negative valuation (default: True)

EXAMPLES:

```
sage: K.<x> = FunctionField(QQ)
sage: R.<y> = K[

```

```
sage: L.<y>=K.extension(y^2 - (x - 1))
```

```
sage: v = K.valuation(0)
sage: w = v.extension(L)
sage: w.reduce(y) # indirect doctest
```

class sage.rings.valuation.limit_valuation.MacLaneLimitValuation\((\text{parent, approximation, G})\)

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 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: 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 , ... ]
```

**element_withvaluation**($s$)

Return an element with valuation $s$.

**extensions**(ring)

Return the extensions of this valuation to ring.

**EXAMPLES:**

```
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_pseudovaluation**()

Return whether this valuation attains $-\infty$.

**EXAMPLES:**

For a Mac Lane limit valuation, this is never the case, so this method always returns False:

```
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.is_negative_pseudovaluation()
False```

**lift**($F$)

Return a lift of $F$ from the residue_ring() to the domain of this valuation.

**EXAMPLES:**

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

(continues on next page)
Discrete Valuations and Discrete Pseudo-Valuations, Release 10.1

(continued from previous page)

```python
u1
sage: w.lift(s) # indirect doctest

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: K = QQ
sage: R.<t> = K[]
sage: L.<t> = K.extension(t^2 + 1)
```
```text
sage: v = QQ.valuation(2)
sage: u = v.extension(L)
sage: u.lower_bound(1024*t + 1024)
10
```
```text
sage: u(1024*t + 1024)
21/2
```

residue_ring()

Return the residue ring of this valuation, which is always a field.

EXAMPLES:

```python
sage: K = QQ
sage: R.<t> = K[]
```
```text
sage: L.<t> = K.extension(t^2 + 1)
sage: v = QQ.valuation(2)
sage: w = v.extension(L)
sage: w.residue_ring()
```
```text
Finite Field of size 2
```

restriction(ring)

Return the restriction of this valuation to ring.

EXAMPLES:

```python
sage: K = QQ
sage: R.<t> = K[]
```
```text
sage: L.<t> = K.extension(t^2 + 1)
sage: v = QQ.valuation(2)
sage: w = v.extension(L)
sage: w._base_valuation.restriction(K)
```
```text
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:
uniformizer()

Return a uniformizing element for this valuation.

EXAMPLES:

```sage
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)
sage: 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:

```sage
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)
sage: u.upper_bound(1024*t + 1024)
21/2
sage: u(1024*t + 1024)
21/2
```

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:

```sage
sage: K.<x> = FunctionField(QQ)
sage: R.<y> = K[]
```

Discrete Valuations and Discrete Pseudo-Valuations, Release 10.1

(continued from previous page)

```
sage: L.<y> = K.extension(y^2 - x)
sage: v = K.valuation(0)
sage: w = v.extension(L); w
(x)-adic valuation
sage: w._base_valuation
[ Gauss valuation induced by (x)-adic valuation, v(y) = 1/2 , ... ]
```

AUTHORS:

- Julian Rüth (2016-11-10): initial version

```python
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:

```
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
```

`lower_bound(x)`

Return an 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: 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.lower_bound(t + 2)
0
sage: u(t + 2)
1
```

`restriction(ring)`

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

EXAMPLES:
Discrete Valuations and Discrete Pseudo-Valuations, Release 10.1

```
sage: K = QQ
sage: R.<t> = K[]
```

```
sage: L.<t> = K.extension(t^2 + 1)
sage: v = valuations.pAdicValuation(QQ, 2)
sage: w = v.extension(L)
sage: 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:

```
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: f = 125*t + 1
```

```
sage: 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:

```
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 + 2) >= 1
```

```
True
```

```
sage: u(t + 2)
1
```

```
class sage.rings.valuation.mapped_valuation.FiniteExtensionFromLimitValuation (parent, approximant, G, approximants)
```

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:

```
sage: K.<x> = FunctionField(QQ)
sage: R.<y> = K[]
```

(continues on next page)
Discrete Valuations and Discrete Pseudo-Valuations, Release 10.1

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

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: 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

element_with_valuation(s)

Return an element with valuation s.

EXAMPLES:

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.element_with_valuation(1)
5

lift(F)

Lift F from the residue_field() of this valuation into its domain.

EXAMPLES:

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:
Discrete Valuations and Discrete Pseudo-Valuations, Release 10.1

```python
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)
```

**residue_ring()**

Return the residue ring of this valuation.

**EXAMPLES:**

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

```python
sage: L.<t> = K.extension(t^2 + 1)
sage: v = valuations.pAdicValuation(QQ, 2)
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: 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)[0]
```

As `_relative_size()` misses the bloated term `x^32`, the following term does not get simplified:

```python
sage: w.simplify(y + x^32)
y + x^32
```

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)
y
```

**uniformizer()**

Return a uniformizing element of this valuation.

**EXAMPLES:**
Discrete Valuations and Discrete Pseudo-Valuations, Release 10.1

```python
sage: K = QQ
sage: R.<t> = K[]
sage: L.<t> = K.extension(t^2 + 1)
sage: v = valuations.pAdicValuation(QQ, 2)
sage: v.extension(L).uniformizer()
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

```python
class sage.rings.valuation.scaled_valuation.ScaledValuationFactory
    Bases: UniqueFactory

    Return a valuation which scales the valuation base by the factor s.

    EXAMPLES:

    sage: 3*ZZ.valuation(2) # indirect doctest
    3 * 2-adic valuation
```

```python
def create_key(base, s):
    Create a key which uniquely identifies a valuation.

def create_object(version, key):
    Create a valuation from key.
```

```python
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:

    sage: v = 3*ZZ.valuation(3); v
    3 * 3-adic valuation
```

```python
extensions(ring)
    Return the extensions of this valuation to ring.

    EXAMPLES:

    sage: v = 3*ZZ.valuation(5)
sage: v.extensions(GaussianIntegers().fraction_field())
[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 \texttt{residue_field()} of this valuation into its domain.

**EXAMPLES:**

```
sage: v = 3^*ZZ.valuation(2)
sage: v.lift(1)
1
```

reduce\((f)\)
Return the reduction of \(f\) in the \texttt{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\((\text{ring})\)
Return the restriction of this valuation to \text{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:**

```
sage: v = 3^*ZZ.valuation(2)
sage: v.uniformizer()
2
```

value_semigroup()
Return the value semigroup of this valuation.

**EXAMPLES:**

```
sage: v2 = QQ.valuation(2)
sage: (2*v2).value_semigroup()
Additive Abelian Semigroup generated by -2, 2
```
### 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:

```sage
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:

```sage
sage: R.<x> = QQ[]
sage: w = valuations.GaussValuation(R, QQ.valuation(2))
sage: v = K.valuation(w); v
2-adic valuation
```

Valuations on a rational function field can then be extended to finite extensions:

```sage
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
# needs sage.rings.function_field
[[ (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].

```python
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:**


sage: K.<x> = FunctionField(QQ)
sage: v = K.valuation(x)
sage: R.<y> = K[]
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]

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

A finite place with residual degree:

sage: w = K.valuation(x^2 + 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

A finite place with residual degree and ramification:

sage: q = L.valuation(x^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:

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

(continues on next page)
### Discrete Valuations and Discrete Pseudo-Valuations

#### restriction

Return the restriction of this valuation to ring.

**EXAMPLES:**

```python
sage: # needs sage.rings.function_field
sage: K.<x> = FunctionField(QQ)
sage: L.<y> = K.extension(y^2 - (x^2 + x + 1))
sage: v = K.valuation(x - 1) # indirect doctest
sage: w = v.extension(L); w
(x - 1)-adic valuation
```

#### scale

Return this valuation scaled by scalar.

**EXAMPLES:**
Discrete valuations on function fields

```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 + x + 1))
sage: v = K.valuation(x - 1)  # indirect doctest
sage: w = v.extension(L)
sage: 3*w
3 * (x - 1)-adic valuation
```

```python
class sage.rings.function_field.valuation.FunctionFieldMappedValuationRelative_base(parent, base_valuation, to_base_valuation_domain, from_base_valuation_domain)

Bases: FunctionFieldMappedValuation_base

A valuation on a function field which relies on a base_valuation 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:

```python
sage: K.<x> = FunctionField(GF(2))
sage: v = K.valuation(1/x); v
Valuation at the infinite place
```

```python
restriction(ring)

Return the restriction of this valuation to ring.

EXAMPLES:

```python
sage: K.<x> = FunctionField(GF(2))
sage: K.valuation(1/x).restriction(GF(2))
Trivial valuation on Finite Field of size 2
```

```python
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
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[]
```

(continues on next page)
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

scale(scalar)

Return this valuation scaled by scalar.

EXAMPLES:

sage: K.<x> = FunctionField(GF(2))
sage: R.<y> = K[]
sage: L.<y> = K.extension(y^2 + y + x^3)  # needs sage.rings.function_field
sage: v = K.valuation(1/x)
sage: w = v.extension(L)  # needs sage.rings.function_field
sage: 3*w
3 * (x)-adic valuation (in Rational function field in x over Finite Field of size 2 after x |--> 1/x)

class sage.rings.function_fieldvaluation.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:

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

See sage.rings.function_field.function_field.FunctionField.valuation() for further examples.

create_key_and_extra_args(domain, prime)

Create a unique key which identifies the valuation given by prime on domain.

create_key_and_extra_args_from_place(domain, generator)

Create a unique key which identifies the valuation at the place specified by generator.

create_key_and_extra_args_from_valuation(domain, valuation)

Create a unique key which identifies the valuation which extends valuation.
create_key_and_extra_args_from_valuation_on_isomorphic_field

Create a unique key which identifies the valuation which is valuation after mapping through to_valuation_domain.

create_object

Create the valuation specified by key.

EXAMPLES:

```sage
K.<x> = FunctionField(QQ)
R.<x> = QQ[]
w = valuations.GaussValuation(R, QQ.valuation(2))
v = K.valuation(w); v  # indirect doctest
2-adic valuation
```

class sage.rings.function_field.valuation.

FunctionFieldValuation_base

Bases: DiscretePseudoValuation

Abstract base class for any discrete (pseudo-)valuation on a function field.

class sage.rings.function_field.valuation.

InducedRationalFunctionFieldValuation_base

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:

```sage
K.<x> = FunctionField(QQ)
v = K.valuation(x^2 + 1)
L.<x> = FunctionField(GaussianIntegers().fraction_field())
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:

```sage
K.<x> = FunctionField(QQ)
v = K.valuation(x)
0
v.lift(0)
0
v.lift(1)
1
```

reduce(f)

Return the reduction of f in residue_ring().

EXAMPLES:
sage: K.<x> = FunctionField(QQ)
sage: v = K.valuation(x^2 + 1)
sage: v.reduce(x)
ul

residue_ring()
Return the residue field of this valuation.

EXAMPLES:

sage: K.<x> = FunctionField(QQ)
sage: K.valuation(x).residue_ring()
Rational Field

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

EXAMPLES:

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

sage: K.<x> = FunctionField(QQ)
sage: v = K.valuation(2)
sage: f = (x + 1)/(x - 1)
As the coefficients of this fraction are small, we do not simplify as this could be very costly in some cases:

sage: v.simplify(f)
(x + 1)/(x - 1)

However, simplification can be forced:

sage: 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
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: FunctionFieldMappedValuationRelative_base, RationalFunctionFieldValuation_base, ClassicalFunctionFieldValuation_base

Valuation of the infinite place of a function field.

EXAMPLES:

```sage
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

Valuation induced by a valuation on the underlying polynomial ring which is non-classical.

EXAMPLES:

```sage
sage: K.<x> = FunctionField(QQ)
sage: v = GaussValuation(QQ['x'], QQ.valuation(2))
sage: w = K.valuation(v); w  # indirect doctest
2-adic valuation
```

residue_ring()

Return the residue field of this valuation.

EXAMPLES:

```sage
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
```

5.12. Discrete valuations on function fields 89
class sage.rings.function_field.valuation.RationalFunctionFieldMappedValuation(parent, base_valuation, to_base_valuation_domain, 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)
```

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[]
sage: w = GaussValuation(R, v).augmentation(x, 1/123)
sage: w = w.extension(K)
sage: w.element_with_valuation(122/123)
  2/x
sage: w.element_with_valuation(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)
5-adic valuation
```

(continues on next page)
These valuations can then, e.g., be used to compute approximate factorizations in the completion of a ring:

\[
sage: \quad v = \text{ZZ.valuation}(2) \\
\text{Sage: } R.\,<x> = \text{ZZ}[x] \\
\text{Sage: } f = x^5 + x^4 + x^3 + x^2 + x - 1 \\
\text{Sage: } v.\,\text{montes_factorization}(f, \text{required\_precision}=20) \\
(x + 676027) \ast (x^4 + 372550*x^3 + 464863*x^2 + 385052*x + 297869)
\]

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].

---

**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
- \( \text{prime} \) – a prime that does not split, a discrete (pseudo-)valuation, a fractional ideal, or \( \text{None} \) (default: \( \text{None} \))

EXAMPLES:

For integers and rational numbers, \( \text{prime} \) is just a prime of the integers:

\[
\text{sage: } \text{valuations.pAdicValuation}(\text{ZZ}, 3) \\
\text{3-adic valuation} \\
\text{sage: } \text{valuations.pAdicValuation}(\text{QQ}, 3) \\
\text{3-adic valuation}
\]

\( \text{prime} \) may be \( \text{None} \) for local rings:

\[
\text{sage: } \text{valuations.pAdicValuation}(\text{Qp}(2)) \\
\text{2-adic valuation} \\
\text{sage: } \text{valuations.pAdicValuation}(\text{Zp}(2)) \\
\text{2-adic valuation}
\]

But it must be specified in all other cases:

\[
\text{sage: } \text{valuations.pAdicValuation}(\text{ZZ}) \\
\text{Traceback (most recent call last):} \\
\text{...} \\
\text{ValueError: prime must be specified for this ring}
\]
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):

```python
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

See also:

- `NumberField_generic.valuation()`, `Order.valuation()`, `pAdicGeneric.valuation()`, `RationalField.valuation()`, `IntegerRing_class.valuation()`.

`create_key_and_extra_args` \(_R, \text{prime}=\text{None, approximants}=\text{None}\)  
Create a unique key identifying the valuation of \(R\) with respect to \(\text{prime}\).

EXAMPLES:

```python
sage: QQ.valuation(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
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: GaussianIntegers().valuation(GaussianIntegers().number_field().fractional_ideal(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: GaussianIntegers().valuation(ZZ.valuation(2)) # indirect doctest
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
```

**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
```

**create_object**(*version, key, **extra_args*)
Create a *p*-adic valuation from *key*.

**EXAMPLES:**
```python
sage: ZZ.valuation(5)  # indirect doctest
5-adic valuation
```

**class** `sage.rings.padics.padic_valuation.pAdicFromLimitValuation`(*parent, approximant, G, approximants*)
Bases: `FiniteExtensionFromLimitValuation`, `pAdicValuation_base`

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
3-adic valuation
```

**extensions**(*ring*)
Return the extensions of this valuation to *ring*.

**EXAMPLES:**
```python
sage: v = GaussianIntegers().valuation(3)
sage: v.extensions(v.domain().fraction_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:**
Discrete Valuations and Discrete Pseudo-Valuations, Release 10.1

sage: ZZ.valuation(3)
3-adic valuation

sage: QQ.valuation(5)
5-adic valuation

For \(p\)-adic rings, \(\cdot p\) has to match the \(p\) of the ring. ::

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

**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
- \(\text{include\_steps}\) – a boolean (default: False); where to include the valuations produced during the process of checking whether \(G\) is totally ramified in the return value
- \(\text{assume\_squarefree}\) – 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: k = Qp(5,4)
sage: v = k.valuation()
sage: R.<x> = k[]
sage: G = x^2 + 1
```
We consider an extension as totally ramified if its ramification index matches the degree. Hence, a trivial extension is totally ramified:

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

However, even if $G$ factors, it might define an unramified extension:

```sage
sage: v.is_unramified(x^2 + 2^2*x + 4)
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:

```
sage: GaussianIntegers().valuation(2).p()
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:

```
sage: v = ZZ.valuation(3)
sage: v.reduce(4)
1
```

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

EXAMPLES:

```
sage: v = GaussianIntegers().valuation(2)
sage: v.restriction(ZZ)
2-adic valuation
```

value_semigroup()
Return the value semigroup of this valuation.

EXAMPLES:

```
sage: v = GaussianIntegers().valuation(2)
sage: v.value_semigroup()
Additive Abelian Semigroup generated by 1/2
```
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
```

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
```

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
```

Unless the precision is very small:

```python
sage: v.inverse(2, 0)
1
```

residue_ring()

Return the residue field of this valuation.

EXAMPLES:

```python
sage: v = ZZ.valuation(3)
sage: 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
Discrete Valuations and Discrete Pseudo-Valuations, Release 10.1

- **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:**

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

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
```

**class** `sage.rings.padics.padic_valuation.pAdicValuation_padic(parent)`

Bases: `pAdicValuation_base`

The $p$-adic valuation of a complete $p$-adic ring.

**INPUT:**
- `R` – a $p$-adic ring

**EXAMPLES:**

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

**element_with_valuation(v)**

Return an element of valuation $v$.

**INPUT:**
- `v` – an element of the `pAdicValuation_base.value_semigroup()` of this valuation

**EXAMPLES:**
```sage
R = Zp(3)
sage: v = R.valuation()
sage: v.element_with_valuation(3)
3^3 + O(3^23)

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 residue_field() to the domain of this valuation.

**INPUT:**
- x – an element of the residue field of this valuation

**EXAMPLES:**
```sage
R = Zp(3)
sage: v = R.valuation()
sage: xbar = v.reduce(R(4))
sage: 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 residue_field().

**EXAMPLES:**
```sage
R = Zp(3)
sage: Zp(3).valuation().reduce(R(4))
1
```

### residue_ring()
Return the residue field of this valuation.

**EXAMPLES:**
```sage
Qq(9, names='a').valuation().residue_ring()
Finite Field in a0 of size 3^2
```

### shift(x, s)
Shift x in its expansion with respect to 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.
Discrete Valuations and Discrete Pseudo-Valuations, Release 10.1

EXAMPLES:

\begin{verbatim}
sage: R = ZpCA(2)
sage: v = R.valuation()
sage: v.shift(R.one(), 1) 2 + O(2^20)
sage: v.shift(R.one(), -1) 0(2^19)

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)
\end{verbatim}

\textbf{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.)

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

EXAMPLES:

\begin{verbatim}
\end{verbatim}

\textbf{uniformizer}()

Return a uniformizer of this valuation.

EXAMPLES:

\begin{verbatim}
\end{verbatim}
CHAPTER SIX

INDICES AND TABLES

• Index
• Module Index
• Search Page
sage.rings.function_field.valuation, 82
sage.rings.padics.padicvaluation, 90
sage.rings.valuation.augmentedvaluation, 52
sage.rings.valuation.developingvaluation, 39
sage.rings.valuation.gaussvaluation, 31
sage.rings.valuation.inductivevaluation, 42
sage.rings.valuation.limitvaluation, 70
sage.rings.valuation.mappedvaluation, 75
sage.rings.valuation.scaledvaluation, 80
sage.rings.valuation.trivialvaluation, 28
sage.rings.valuationvaluation, 12
sage.rings.valuationvaluation_space, 19
sage.rings.valuationvaluation_group, 9
A
augmentation() (sage.rings.valuation.inductive_valuation.NonFinalInductiveValuation method), 47
augmentation_chain() (sage.rings.valuation.augmented_valuation.AugmentedValuation_base method), 54
augmentation_chain() (sage.rings.valuation.gauss_valuation.GaussValuation_generic method), 33
augmentation_chain() (sage.rings.valuation.inductive_valuation.InductiveValuation method), 43
AugmentedValuation_base (class in sage.rings.valuation.augmented_valuation), 53
AugmentedValuationFactory (class in sage.rings.valuation.augmented_valuation), 53

C
change_domain() (sage.rings.padic.padic_valuation.pAdicValuation_base method), 94
change_domain() (sage.rings.valuation.augmented_valuation.AugmentedValuation_base method), 55
change_domain() (sage.rings.valuation.gauss_valuation.GaussValuation_generic method), 33
change_domain() (sage.rings.valuation.inductive_valuation.InductiveValuation method), 46
change_domain() (sage.rings.valuation.valuation_space.DiscretePseudoValuationSpace.ElementMethods method), 21
ClassicalFunctionFieldValuation_base (class in sage.rings.function_field.valuation), 82
coefficients() (sage.rings.valuation.developing_valuation.DevelopingValuation method), 40
create_key() (sage.rings.valuation.augmented_valuation.AugmentedValuationFactory method), 53
create_key() (sage.rings.valuation.gauss_valuation.GaussValuationFactory method), 32
create_key() (sage.rings.valuation.limit_valuation.LimitValuationFactory method), 71
create_key() (sage.rings.valuation.scaled_valuation.ScaledValuationFactory method), 80
create_key() (sage.rings.valuation.trivial_valuation.TrivialValuationFactory method), 31
create_key_and_extra_args() (sage.rings.function_field.valuation.FunctionFieldValuationFactory method), 86
create_key_and_extra_args() (sage.rings.padic.padic_valuation.PadicValuationFactory method), 92
create_key_and_extra_args_for_number_field() (sage.rings.padic.padic_valuation.PadicValuationFactory method), 92
create_key_and_extra_args_for_number_field_from_ideal() (sage.rings.padic.padic_valuation.PadicValuationFactory method), 92
create_key_and_extra_args_for_number_field_from_valuation() (sage.rings.padic.padic_valuation.PadicValuationFactory method), 92
create_key_and_extra_args_from_place() (sage.rings.function_field.valuation.FunctionFieldValuationFactory method), 86
create_key_and_extra_args_from_valuation() (sage.rings.function_field.valuation.FunctionFieldValuationFactory method), 86
create_key_and_extra_args_from_valuation_on_isomorphic_field() (sage.rings.function_field.valuation.FunctionFieldValuationFactory method), 86
create_key_for_integers() (sage.rings.padic.padic_valuation.PadicValuationFactory method), 92
create_key_for_local_ring() (sage.rings.padic.padic_valuation.PadicValuationFactory method), 93
create_key_for_local_ring() (sage.rings.valuation.augmented_valuation.AugmentedValuationFactory method), 53
create_key_for_local_ring() (sage.rings.valuation.gauss_valuation.GaussValuationFactory method), 32
create_key_for_local_ring() (sage.rings.valuation.limit_valuation.LimitValuationFactory method), 71
create_key_for_local_ring() (sage.rings.valuation.scaled_valuation.ScaledValuationFactory method), 80
create_key_for_local_ring() (sage.rings.valuation.trivial_valuation.TrivialValuationFactory method), 31
create_object() (sage.rings.function_field.valuation.FunctionFieldValuationFactory method), 87
create_object() (sage.rings.padic.padic_valuation.PadicValuationFactory method), 93
create_object() (sage.rings.padic.padic_valuation.PadicValuationFactory method), 93
create_object() (sage.rings.valuation.trivial_valuation.TrivialDiscreteValuation_base.m
ethod), 31

denominator() (sage.rings.valuation.value_group.DiscreteValueGroup_m
ethod), 10

DevelopingValuation (class in sage.rings.valuation.developing_valuation), 39

DiscreteFunctionFieldValuation_base (class in sage.rings.function_field.valuation), 82

DiscretePseudovaluation (class in sage.rings.valuation.pseudovaluation), 13

DiscretePseudovaluationSpace (class in sage.rings.valuation.pseudovaluation_space), 20

DiscretePseudovaluationSpace.ElementMethods (class in sage.rings.valuation.pseudovaluation_space), 20

Discretevaluation (class in sage.rings.valuation.pseudovaluation), 13

DiscretevaluationCodomain (class in sage.rings.valuation.value_group), 9

DiscretevaluationGroup (class in sage.rings.valuation.value_group), 9

DiscretevaluationSemigroup (class in sage.rings.valuation.value_group), 11

effective_degree() (sage.rings.valuation.developing_valuation.Developingvaluation
method), 40

element\_with\_valuation() (sage.rings.function\_field.valuation.RationalFunctionFieldvaluation\_base
method), 90

element\_with\_valuation() (sage.rings.padics.padic\_valuation.pAdicLimitvaluation\_base
method), 98

element\_with\_valuation() (sage.rings.valuation.augmented_valuation.Augmentedvaluation
base
method), 55

element\_with\_valuation() (sage.rings.valuation.gauss\_valuation.Gaussvaluation\_generic
method), 33

element\_with\_valuation() (sage.rings.valuation.inductive\_valuation.Inductivevaluation
method), 43

element\_with\_valuation() (sage.rings.valuation.limit\_valuation.MacLaneLimitvaluation\_base
method), 73

equivalence\_decomposition() (sage.rings.valuation.inductive\_valuation.NonFinalInductivevaluation
method), 47

equivalence\_reciprocal() (sage.rings.valuation.inductive\_valuation.Inductivevaluation
method), 43

equivalence\_unit() (sage.rings.valuation.augmented\_valuation.Augmentedvaluation
method), 56

equivalence\_unit() (sage.rings.valuation.gauss\_valuation.Gaussvaluation
method), 33

equivalence\_unit() (sage.rings.valuation.inductive\_valuation.Inductivevaluation
method), 44

extension() (sage.rings.valuation.pseudovaluation, 21

extensions() (sage.rings.function\_field.valuation.Discrete\_FunctionFieldvaluation
method), 82

extensions() (sage.rings.function\_field.valuation.Induced\_Rational\_Function\_Fieldvaluation
method), 87

extensions() (sage.rings.padics.padic\_valuation.pAdic\_From\_Limitvaluation\_base
method), 93

extensions() (sage.rings.padics.padic\_valuation.pAdic\_valuation\_base
method), 94

extensions() (sage.rings.valuation.augmented\_valuation.Augmentedvaluation
method), 56

extensions() (sage.rings.valuation.gauss\_valuation.Gaussvaluation\_generic
method), 34

extensions() (sage.rings.valuation.inductive\_valuation.Finite\_Inductivevaluation
method), 72

extensions() (sage.rings.valuation.limit\_valuation.MacLane\_Limitvaluation\_base
method), 73

extensions() (sage.rings.valuation.scaled\_valuation.Scaledvaluation\_generic
method), 80

extensions() (sage.rings.valuation.trivial\_valuation.Trivial\_Discretevaluation
method), 30

extensions() (sage.rings.valuation.pseudovaluation, 21
monic_integral_model() (sage.rings.valuation.augmented_valuation.AugmentedValuationRing_field, 57)
monic_integral_model() (sage.rings.valuation.gauss_valuation.GaussValuationRing_field, 58)
monic_integral_model() (sage.rings.valuation.inductive_valuation.InductiveValuation, 45)
montes_factorization() (sage.rings.valuation.valuation.DiscreteValuation, 17)
mu() (sage.rings.valuation.inductive_valuation.InductiveValuation, 46)

NegativeInfiniteDiscretePseudoValuation (class in sage.rings.valuation, 18)
newton_polygon() (sage.rings.valuation.developing_valuation.DevelopingValuation, 36)
NonClassicalRationalFunctionFieldValuation (class in sage.rings.function_field.valuation, 89)
NonFinalAugmentedValuation (class in sage.rings.valuation.augmented_valuation, 66)
NonFinalFiniteAugmentedValuation (class in sage.rings.valuation.augmented_valuation, 69)
NonFinalInductiveValuation (class in sage.rings.valuation.inductive_valuation, 46)
numerator() (sage.rings.valuation.value_group.DiscreteValuation, 89)

P
p() (sage.rings.padics.padic_valuation.pAdicValuation_base, 96)
pAdicFromLimitValuation (class sage.rings.padics.padic_valuation, 93)
pAdicValuation_base (class sage.rings.padics.padic_valuation, 93)
pAdicValuation_int (class sage.rings.padics.padic_valuation, 96)
pAdicValuation_padic (class sage.rings.padics.padic_valuation, 98)

PadicValuationFactory (class sage.rings.padics.padic_valuation, 91)
phi() (sage.rings.valuation.developing_valuation.DevelopingValuation, 41)
psi() (sage.rings.valuation.augmented_valuation.AugmentedValuationRing_field, 58)

R
RationalFunctionFieldMappedValuation (class in sage.rings.function_field.valuation, 89)