1 Power Series Rings 1
2 Power Series 11
3 Power Series Methods 31
4 Power series implemented using PARI 37
5 Multivariate Power Series Rings 43
6 Multivariate Power Series 51
7 Laurent Series Rings 67
8 Laurent Series 73
9 Indices and Tables 85
Python Module Index 87
Index 89
Power series rings are constructed in the standard Sage fashion. See also *Multivariate Power Series Rings*.

### EXAMPLES:

Construct rings and elements:

```
sage: R.<t> = PowerSeriesRing(QQ)
sage: R.random_element(6)  # random
-4 - 1/2*t^2 - 1/95*t^3 + 1/2*t^4 - 12*t^5 + O(t^6)
```

```
sage: R.<t,u,v> = PowerSeriesRing(QQ); R
Multivariate Power Series Ring in t, u, v over Rational Field
sage: p = -t + 1/2*t^3*u - 1/4*t^4*u + 2/3*v^5 + R.O(6); p
-t + 1/2*t^3*u - 1/4*t^4*u + 2/3*v^5 + O(t, u, v)^6
sage: p in R
True
```

The default precision is specified at construction, but does not bound the precision of created elements.

```
sage: R.<t> = PowerSeriesRing(QQ, default_prec=5)
sage: R.random_element(6)  # random
1/2 - 1/4*t + 2/3*t^2 - 5/2*t^3 + 2/3*t^5 + O(t^6)
```

Construct univariate power series from a list of coefficients:

```
sage: S = R([1, 3, 5, 7]); S
1 + 3*t + 5*t^2 + 7*t^3
```

An iterated example:

```
sage: R.<t> = PowerSeriesRing(ZZ)
sage: S.<t2> = PowerSeriesRing(R)
sage: S
Power Series Ring in t2 over Power Series Ring in t over Integer Ring
sage: S.base_ring()
Power Series Ring in t over Integer Ring
```

Sage can compute with power series over the symbolic ring.

```
sage: K.<t> = PowerSeriesRing(SR, default_prec=5)
sage: a, b, c = var('a,b,c')
sage: f = a + b*t + c*t^2 + O(t^3)
sage: f*f
a^2 + 2*a*b*t + (b^2 + 2*a*c)*t^2 + O(t^3)
```

(continues on next page)
Elements are first coerced to constants in `base_ring`, then coerced into the `PowerSeriesRing`:

```
sage: R.<t> = PowerSeriesRing(ZZ)
sage: f = Mod(2, 3) * t; (f, f.parent())
(2*t, Power Series Ring in t over Ring of integers modulo 3)
```

We make a sparse power series.

```
sage: R.<x> = PowerSeriesRing(QQ, sparse=True); R
Sparse Power Series Ring in x over Rational Field
sage: f = 1 + x^1000000
sage: g = f*f
sage: g.degree()
2000000
```

We make a sparse Laurent series from a power series generator:

```
sage: R.<t> = PowerSeriesRing(QQ, sparse=True)
sage: latex(-2/3*(1/t^3) + 1/t + 3/5*t^2 + O(t^5))
\frac{-\frac{2}{3}}{t^{3}} + \frac{1}{t} + \frac{3}{5}t^{2} + O(t^{5})
sage: S = parent(1/t); S
Sparse Laurent Series Ring in t over Rational Field
```

AUTHORS:

- William Stein: the code
- Jeremy Cho (2006-05-17): some examples (above)
- Niles Johnson (2010-09): implement multivariate power series
- Simon King (2012-08): use category and coercion framework, trac ticket #13412

```
sage: R.<x> = PowerSeriesRing(QQ, sparse=True)
sage: R is loads(dumps(R))
True
sage: TestSuite(R).run()
sage: M = PowerSeriesRing(QQ, 't,u,v,w', default_prec=20)
sage: M is loads(dumps(M))
True
sage: TestSuite(M).run()
```

Create a univariate or multivariate power series ring over a given (commutative) base ring.

**INPUT:**

- `base_ring` - a commutative ring
- `name, names` - name(s) of the indeterminate
• **default_prec** - the default precision used if an exact object must be changed to an approximate object in order to do an arithmetic operation. If left as None, it will be set to the global default (20) in the univariate case, and 12 in the multivariate case.

• **sparse** - (default: False) whether power series are represented as sparse objects.

• **order** - (default: negdeglex) term ordering, for multivariate case

• **num_gens** - number of generators, for multivariate case

There is a unique power series ring over each base ring with given variable name. Two power series over the same base ring with different variable names are not equal or isomorphic.

**EXAMPLES (Univariate):**

```sage
sage: R = PowerSeriesRing(QQ, 'x'); R
Power Series Ring in x over Rational Field

sage: S = PowerSeriesRing(QQ, 'y'); S
Power Series Ring in y over Rational Field

sage: R = PowerSeriesRing(QQ, 10)
Traceback (most recent call last):
...  
ValueError: variable name '10' does not start with a letter

sage: S = PowerSeriesRing(QQ, 'x', default_prec = 15); S
Power Series Ring in x over Rational Field
sage: S.default_prec()
15
```

**EXAMPLES (Multivariate) See also *Multivariate Power Series Rings*:**

```sage
sage: R = PowerSeriesRing(QQ, 't,u,v'); R
Multivariate Power Series Ring in t, u, v over Rational Field

sage: N = PowerSeriesRing(QQ,'w',num_gens=5); N
Multivariate Power Series Ring in w0, w1, w2, w3, w4 over Rational Field

Number of generators can be specified before variable name without using keyword:

```sage
sage: M = PowerSeriesRing(QQ,4,'k'); M
Multivariate Power Series Ring in k0, k1, k2, k3 over Rational Field
```

Multivariate power series can be constructed using angle bracket or double square bracket notation:

```sage
sage: R.<t,u,v> = PowerSeriesRing(QQ, 't,u,v'); R
Multivariate Power Series Ring in t, u, v over Rational Field
sage: ZZ[['s,t,u']]
Multivariate Power Series Ring in s, t, u over Integer Ring
```

Sparse multivariate power series ring:

```sage
sage: M = PowerSeriesRing(QQ,4,'k',sparse=True); M
Sparse Multivariate Power Series Ring in k0, k1, k2, k3 over Rational Field
```

Power series ring over polynomial ring:
Power series ring over finite field:

```python
sage: S = PowerSeriesRing(GF(65537),'x,y'); S
Multivariate Power Series Ring in x, y over Finite Field of size 65537
```

Power series ring with many variables:

```python
sage: R = PowerSeriesRing(ZZ, ['x%s%s' % p for p in primes(100)]); R
Multivariate Power Series Ring in x2, x3, x5, x7, x11, x13, x17, x19, x23, x29, x31, x37, x41, x43, x47, x53, x59, x61, x67, x71, x73, x79, x83, x89, x97 over Integer Ring
```

- Use `inject_variables()` to make the variables available for interactive use.

```python
sage: R.inject_variables()
Defining x2, x3, x5, x7, x11, x13, x17, x19, x23, x29, x31, x37, x41, x43, x47, x53, x59, x61, x67, x71, x73, x79, x83, x89, x97

sage: f = x47 + 3*x11*x29 - x19 + R.O(3)
sage: f in R
True
```

Variable ordering determines how series are displayed:

```python
sage: T.<a,b> = PowerSeriesRing(ZZ, order='deglex'); T
Multivariate Power Series Ring in a, b over Integer Ring

sage: T.term_order()
Degree lexicographic term order

sage: p = - 2*b^6 + a^5*b^2 + a^7 - b^2 - a*b^3 + T.O(9); p
a^7 + a^5*b^2 - 2*b^6 - a*b^3 - b^2 + O(a, b)^9

sage: U = PowerSeriesRing(ZZ,'a,b',order='negdeglex'); U
Multivariate Power Series Ring in a, b over Integer Ring

sage: U.term_order()
Negative degree lexicographic term order

sage: U(p)
-b^2 - a*b^3 - 2*b^6 + a^7 + a^5*b^2 + O(a, b)^9
```

See also:

- `sage.misc.defaults.set_series_precision()`

```python
class sage.rings.power_series_ring.PowerSeriesRing_domain(base_ring, name=None, default_prec=None, sparse=False, implementation=None, category=None)
```

Bases: `sage.rings.power_series_ring.PowerSeriesRing_generic`, `sage.rings.ring.IntegralDomain`
A power series ring.

**base_extend** \((R)\)

Return the power series ring over \(R\) in the same variable as self, assuming there is a canonical coerce map from the base ring of self to \(R\).

**change_ring** \((R)\)

Return the power series ring over \(R\) in the same variable as self.

**change_var** \((var)\)

Return the power series ring in variable \(var\) over the same base ring.

**characteristic**()

Return the characteristic of this power series ring, which is the same as the characteristic of the base ring of the power series ring.
construction()
Return the functorial construction of self, namely, completion of the univariate polynomial ring with respect to the indeterminate (to a given precision).

EXAMPLES:

```
sage: R = PowerSeriesRing(ZZ, 'x')
sage: c, S = R.construction(); S
Univariate Polynomial Ring in x over Integer Ring
sage: R == c(S)
True
```

gen(n=0)
Return the generator of this power series ring.

EXAMPLES:

```
sage: R.<t> = PowerSeriesRing(ZZ)
sage: R.gen()
t
sage: R.gen(3)
Traceback (most recent call last):
... IndexError: generator n>0 not defined
```

is_dense()
EXAMPLES:

```
sage: R.<t> = PowerSeriesRing(ZZ)
sage: t.is_dense()
True
sage: R.<t> = PowerSeriesRing(ZZ, sparse=True)
sage: t.is_dense()
False
```

is_exact()
Return False since the ring of power series over any ring is not exact.

EXAMPLES:

```
sage: R.<t> = PowerSeriesRing(ZZ)
sage: R.is_exact()
False
```

is_field(proof=True)
Return False since the ring of power series over any ring is never a field.

EXAMPLES:
sage: R.<t> = PowerSeriesRing(ZZ)
sage: R.is_field()
False

**is_finite()**

Return False since the ring of power series over any ring is never finite.

**EXAMPLES:**

```python
sage: R.<t> = PowerSeriesRing(ZZ)
sage: R.is_finite()
False
```

**is_sparse()**

**EXAMPLES:**

```python
sage: R.<t> = PowerSeriesRing(ZZ)
sage: t.is_sparse()
False
sage: R.<t> = PowerSeriesRing(ZZ, sparse=True)
sage: t.is_sparse()
True
```

**laurent_series_ring()**

If this is the power series ring $R[[t]]$, return the Laurent series ring $R((t))$.

**EXAMPLES:**

```python
sage: R.<t> = PowerSeriesRing(ZZ,default_prec=5)
sage: S = R.laurent_series_ring(); S
Laurent Series Ring in t over Integer Ring
sage: S.default_prec()
5
sage: f = 1+t; g=1/f; g
1 - t + t^2 - t^3 + t^4 + O(t^5)
```

**ngens()**

Return the number of generators of this power series ring.

This is always 1.

**EXAMPLES:**

```python
sage: R.<t> = ZZ[[[]]
sage: R.ngens()
1
```

**random_element**(prec=None, *args, **kwds)**

Return a random power series.

**INPUT:**

- prec - Integer specifying precision of output (default: default precision of self)
- *args, **kwds - Passed on to the random_element method for the base ring

**OUTPUT:**

- Power series with precision `prec` whose coefficients are random elements from the base ring, randomized subject to the arguments *args and **kwds
ALGORITHM:
Call the random_element method on the underlying polynomial ring.

EXAMPLES:

```
sage: R.<t> = PowerSeriesRing(QQ)
sage: R.random_element(5) # random
-4 - 1/2*t^2 - 1/95*t^3 + 1/2*t^4 + O(t^5)
sage: R.random_element(10) # random
-1/2 + 2*t - 2/7*t^2 - 25*t^3 - 4*t^4 + 2*t^5 - 4*t^7 - 1/3*t^8 - t^9 + O(t^10)
```

If given no argument, random_element uses default precision of self:

```
sage: T = PowerSeriesRing(ZZ,'t')
sage: T.default_prec()
20
sage: T.random_element() # random
4 + 2*t - t^2 - t^3 + 2*t^4 + t^5 + t^6 - 2*t^7 - t^8 - t^9 + t^11 - 6*t^12 + ...
→ 2*t^14 + 2*t^16 - t^17 - 3*t^18 + O(t^20)
sage: S = PowerSeriesRing(ZZ,'t', default_prec=4)
sage: S.random_element() # random
2 - t - 5*t^2 + O(t^3)
```

Further arguments are passed to the underlying base ring (trac ticket #9481):

```
sage: SZ = PowerSeriesRing(ZZ,'v')
sage: SQ = PowerSeriesRing(QQ,'v')
sage: SR = PowerSeriesRing(RR,'v')

sage: SZ.random_element(x=4, y=6) # random
4 + 5*v + 5*v^2 + 5*v^3 + 4*v^4 + 5*v^5 + 5*v^6 + 5*v^7 + 4*v^8 + 5*v^9 + 4*v^10 + 4*v^11 + 5*v^12 + 5*v^13 + 5*v^14 + 5*v^15 + 5*v^16 + 5*v^17 + 4*v^18 + 5*v^19 + O(v^20)
sage: SQ.random_element(x=3, y=6) # random
5 + 4*v + 5*v^2 + 0(v^3)
sage: SR.random_element(x=3, num_bound=3, den_bound=100) # random
1/87 - 3/70*v - 3/44*v^2 + O(v^3)
sage: SR.random_element(3, max=10, min=-10) # random
2.85948321262904 - 9.73071330911226*v - 6.60414378519265*v^2 + O(v^3)
```

residue_field()
Return the residue field of this power series ring.

EXAMPLES:

```
sage: R.<x> = PowerSeriesRing(GF(17))
sage: R.residue_field()
Finite Field of size 17
sage: R.<x> = PowerSeriesRing(Zp(5))
sage: R.residue_field()
Finite Field of size 5
```

uniformizer()
Return a uniformizer of this power series ring if it is a discrete valuation ring (i.e., if the base ring is actually a field). Otherwise, an error is raised.

EXAMPLES:
sage: R.<t> = PowerSeriesRing(QQ)
sage: R.uniformizer()
t
sage: R.<t> = PowerSeriesRing(ZZ)
sage: R.uniformizer()
Traceback (most recent call last):
  ...
TypeError: The base ring is not a field

variable_names_recursive(depth=None)
Return the list of variable names of this and its base rings.

EXAMPLES:

sage: R = QQ[['x']][['y']][['z']]
sage: R.variable_names_recursive()
('x', 'y', 'z')
sage: R.variable_names_recursive(2)
('y', 'z')

class sage.rings.power_series_ring.PowerSeriesRing_over_field(base_ring, name=None, default_prec=None, sparse=False, implementation=None, category=None)

Bases: sage.rings.power_series_ring.PowerSeriesRing_domain

fraction_field()
Return the fraction field of this power series ring, which is defined since this is over a field.

This fraction field is just the Laurent series ring over the base field.

EXAMPLES:

sage: R.<t> = PowerSeriesRing(GF(7))
sage: R.fraction_field()
Laurent Series Ring in t over Finite Field of size 7

sage.rings.power_series_ring.is_PowerSeriesRing(R)
Return True if this is a univariate power series ring. This is in keeping with the behavior of is_PolynomialRing versus is_MPolynomialRing.

EXAMPLES:

sage: from sage.rings.power_series_ring import is_PowerSeriesRing
sage: is_PowerSeriesRing(10)
False
sage: is_PowerSeriesRing(QQ[['x']])
True

sage.rings.power_series_ring.unpickle_power_series_ring_v0(base_ring, name, default_prec, sparse)
Unpickle (deserialize) a univariate power series ring according to the given inputs.
EXAMPLES:

```python
sage: P.<x> = PowerSeriesRing(QQ)
sage: loads(dumps(P)) == P  # indirect doctest
True
```
Sage provides an implementation of dense and sparse power series over any Sage base ring. This is the base class of the implementations of univariate and multivariate power series ring elements in Sage (see also *Power Series Methods, Multivariate Power Series*).

**AUTHORS:**
- William Stein
- David Harvey (2006-09-11): added `solve_linear_de` method
- Simon King (2012-08): use category and coercion framework, trac ticket #13412

**EXAMPLES:**

```python
sage: R.<x> = PowerSeriesRing(ZZ)
sage: TestSuite(R).run()
sage: R([1,2,3])
1 + 2*x + 3*x^2
sage: R([1,2,3], 10)
1 + 2*x + 3*x^2 + O(x^10)
sage: f = 1 + 2*x - 3*x^3 + O(x^4); f
1 + 2*x - 3*x^3 + O(x^4)
sage: f^10
1 + 20*x + 180*x^2 + 930*x^3 + O(x^4)
sage: g = 1/f; g
1 - 2*x + 4*x^2 - 5*x^3 + O(x^4)
sage: g * f
1 + O(x^4)
```

In Python (as opposed to Sage) create the power series ring and its generator as follows:

```python
sage: R = PowerSeriesRing(ZZ, 'x')
sage: x = R.gen()
sage: parent(x)
Power Series Ring in x over Integer Ring
```

**EXAMPLES:**

This example illustrates that coercion for power series rings is consistent with coercion for polynomial rings.

```python
sage: poly_ring1.<gen1> = PolynomialRing(QQ)
sage: poly_ring2.<gen2> = PolynomialRing(QQ)
sage: huge_ring.<x> = PolynomialRing(poly_ring1)
```
The generator of the first ring gets coerced in as itself, since it is the base ring.

```
sage: huge_ring(gen1)
gen1
```

The generator of the second ring gets mapped via the natural map sending one generator to the other.

```
sage: huge_ring(gen2)
x
```

With power series the behavior is the same.

```
sage: power_ring1.<gen1> = PowerSeriesRing(QQ)
sage: power_ring2.<gen2> = PowerSeriesRing(QQ)
sage: huge_power_ring.<x> = PowerSeriesRing(power_ring1)
sage: huge_power_ring(gen1)
gen1
sage: huge_power_ring(gen2)
x
```

```
class sage.rings.power_series_ring_element.PowerSeries
    Bases: sage.structure.element.AlgebraElement

    A power series. Base class of univariate and multivariate power series. The following methods are available with both types of objects.

    O(prec)
    Return this series plus $O(x^{\text{prec}})$. Does not change self.

    EXAMPLES:

    ```
sage: R.<x> = PowerSeriesRing(ZZ)
sage: p = 1 + x^2 + x^10; p
1 + x^2 + x^10
sage: p.O(15)
1 + x^2 + x^10 + O(x^15)
sage: p.O(5)
1 + x^2 + O(x^5)
sage: p.O(-5)
Traceback (most recent call last):
  ...
ValueError: prec (= -5) must be non-negative
```

```
V(n)
If $f = \sum a_m x^m$, then this function returns $\sum a_m x^{nm}$.

    EXAMPLES:

    ```
sage: R.<x> = PowerSeriesRing(ZZ)
sage: p = 1 + x^2 + x^10; p
1 + x^2 + x^10
sage: p.V(3)
1 + x^6 + x^30
sage: (p+O(x^20)).V(3)
1 + x^6 + x^30 + O(x^60)
```

```
add_bigoh(prec)
Return the power series of precision at most $\text{prec}$ got by adding $O(q^{\text{prec}})$ to $f$, where $q$ is the variable.

    EXAMPLES:
```
base_extend($R$)

Return a copy of this power series but with coefficients in $R$.

The following coercion uses base_extend implicitly:

```
sage: R.<t> = ZZ[['t']]
sage: (t - t^2) * Mod(1, 3)
t + 2*t^2
```

base_ring()

Return the base ring that this power series is defined over.

**EXAMPLES:**

```
sage: R.<t> = GF(49,'alpha')[['t']]
sage: (t^2 + O(t^3)).base_ring()
Finite Field in alpha of size 7^2
```

change_ring($R$)

Change if possible the coefficients of self to lie in $R$.

**EXAMPLES:**

```
sage: R.<T> = QQ[['T']]; R
Power Series Ring in T over Rational Field
sage: f = 1 - 1/2*T + 1/3*T^2 + O(T^3)
sage: f.base_extend(GF(5))
Traceback (most recent call last):
  ...TypeError: no base extension defined
sage: f.change_ring(GF(5))
1 + 2*T + 2*T^2 + O(T^3)
sage: f.change_ring(GF(3))
Traceback (most recent call last):
  ...
ZeroDivisionError: inverse of Mod(0, 3) does not exist
```

We can only change the ring if there is a __call__ coercion defined. The following succeeds because $\mathbb{Z}[[K(4)]]$ is defined.

```
sage: K.<a> = NumberField(cyclotomic_polynomial(3), 'a')
sage: R.<t> = K[['t']]
sage: (4*t).change_ring(ZZ)
4*t
```

This does not succeed because $\mathbb{Z}[[K(a+1)]]$ is not defined.

```
sage: K.<a> = NumberField(cyclotomic_polynomial(3), 'a')
sage: R.<t> = K[['t']]
sage: ((a+1)*t).change_ring(ZZ)
```

(continues on next page)
Traceback (most recent call last):
...
TypeError: Unable to coerce a + 1 to an integer

coefficients()
Return the nonzero coefficients of self.

EXAMPLES:

```sage
sage: R.<t> = PowerSeriesRing(QQ)
sage: f = t + t^2 - 10/3*t^3
sage: f.coefficients()
[1, 1, -10/3]
```

common_prec(f)
Return minimum precision of \( f \) and self.

EXAMPLES:

```sage
sage: R.<t> = PowerSeriesRing(QQ)

sage: f = t + t^2 + O(t^3)
sage: g = t + t^3 + t^4 + O(t^4)
sage: f.common_prec(g)
3
sage: g.common_prec(f)
3
sage: f = t + t^2 + O(t^3)
sage: g = t^2
sage: f.common_prec(g)
3
sage: g.common_prec(f)
3
sage: f = t + t^2
sage: f = t^2
sage: f.common_prec(g)
+Infinity
```

cos(prec='infinity')
Apply cos to the formal power series.

INPUT:

- \texttt{prec} - Integer or \texttt{infinity}. The degree to truncate the result to.

OUTPUT:

A new power series.

EXAMPLES:

For one variable:

```sage
sage: t = PowerSeriesRing(QQ, 't').gen()
sage: f = (t + t**2).O(4)
```
sage: cos(f)
1 - 1/2*t^2 - t^3 + O(t^4)

For several variables:

sage: T.<a,b> = PowerSeriesRing(ZZ,2)
sage: f = a + b + a*b + T.O(3)
sage: cos(f)
1 - 1/2*a^2 - a*b - 1/2*b^2 + O(a, b)^3

If the power series has a non-zero constant coefficient \( c \), one raises an error:

sage: g = 2+f
sage: cos(g)
Traceback (most recent call last):
  ...
ValueError: Can only apply cos to formal power series with zero constant term.

If no precision is specified, the default precision is used:

sage: T.default_prec()
12
sage: cos(a)
1 - 1/2*a^2 + 1/24*a^4 - 1/720*a^6 + 1/40320*a^8 - 1/3628800*a^10 + O(a, b)^12

degree()

Return the degree of this power series, which is by definition the degree of the underlying polynomial.

EXAMPLES:

sage: R.<t> = PowerSeriesRing(QQ, sparse=True)
sage: f = t^100000 + O(t^1000000)
sage: f.degree()
100000

derivative(*args)

The formal derivative of this power series, with respect to variables supplied in args.

Multiple variables and iteration counts may be supplied; see documentation for the global derivative() function for more details.

See also:

_derivative()

EXAMPLES:

sage: R.<x> = PowerSeriesRing(QQ)
sage: g = -x + x^2/2 - x^4 + O(x^6)
sage: g.derivative()
-1 + x - 4*x^3 + O(x^5)
\text{sage: } g . \text{derivative}(x)
-1 + x - 4*x^3 + O(x^5)
\text{sage: } g . \text{derivative}(x, x)
1 - 12*x^2 + O(x^4)
\text{sage: } g . \text{derivative}(x, 2)
1 - 12*x^2 + O(x^4)

\textbf{egf_to_ogf()}  
Returns the ordinary generating function power series, assuming self is an exponential generating function power series.

This function is known as \texttt{serlaplace} in \texttt{PARI/GP}.

\textbf{EXAMPLES:}
\begin{verbatim}
\texttt{sage: R.<t> = PowerSeriesRing(QQ)
\texttt{sage: f = t + t^2/factorial(2) + 2*t^3/factorial(3)
\texttt{sage: f.egf_to_ogf()}
t + t^2 + 2*t^3
\end{verbatim}

\textbf{exp (prec=None)}  
Return exp of this power series to the indicated precision.

\textbf{INPUT:}
- \texttt{prec} - integer; default is self.parent().default_prec

\textbf{ALGORITHM:} See solve_linear_de().

\textbf{Note:}
- Screwy things can happen if the coefficient ring is not a field of characteristic zero. See solve_linear_de().

\textbf{AUTHORS:}
- David Harvey (2006-09-08): rewrote to use simplest possible “lazy” algorithm.
- David Harvey (2006-09-10): rewrote to use divide-and-conquer strategy.
- David Harvey (2006-09-11): factored functionality out to solve_linear_de().
- Sourav Sen Gupta, David Harvey (2008-11): handle constant term

\textbf{EXAMPLES:}
\begin{verbatim}
\texttt{sage: R.<t> = PowerSeriesRing(QQ, default_prec=10)
\texttt{Check that } \texttt{exp(t)} \texttt{is, well, exp(t):}
\texttt{(t + O(t^10)).exp()}
1 + t + 1/2*t^2 + 1/6*t^3 + 1/24*t^4 + 1/120*t^5 + 1/720*t^6 + 1/5040*t^7 + 1/
\text{-40320}*t^8 + 1/362880*t^9 + O(t^10)
\texttt{Check that } \texttt{exp(log(1 + t))} \texttt{is } 1 + t:
\texttt{((sum([-(-t)^n/n for n in range(1, 10)]) + O(t^10)).exp())}
1 + t + O(t^10)
\end{verbatim}
Check that $\exp(2t + t^2 - t^5)$ is whatever it is:

```
sage: (2*t + t^2 - t^5 + O(t^10)).exp()
sage: 1 + 2*t + 3*t^2 + 10/3*t^3 + 19/6*t^4 + 8/5*t^5 - 7/90*t^6 - 538/315*t^7 -
  \rightarrow 425/168*t^8 - 30629/11340*t^9 + O(t^10)
```

Check requesting lower precision:

```
sage: (t + t^2 - t^5 + O(t^10)).exp(5)
sage: 1 + t + 3/2*t^2 + 7/6*t^3 + 25/24*t^4 + O(t^5)
```

Can’t get more precision than the input:

```
sage: (t + t^2 + O(t^3)).exp(10)
sage: 1 + t + 3/2*t^2 + O(t^3)
```

Check some boundary cases:

```
sage: (t + O(t^2)).exp(1)
sage: 1 + O(t)
sage: (t + O(t^2)).exp(0)
sage: O(t^0)
```

Handle nonzero constant term (fixes trac ticket #4477):

```
sage: R.<x> = PowerSeriesRing(RR)
sage: (1 + x + x^2 + O(x^3)).exp()
sage: 2.71828182845905 + 2.71828182845905*x + 4.07742274268857*x^2 + O(x^3)
```

```
sage: R.<x> = PowerSeriesRing(ZZ)
sage: (1 + x + O(x^2)).exp()
Traceback (most recent call last):
...
ArithmeticError: exponential of constant term does not belong to coefficient ring (consider working in a larger ring)
```

```
sage: R.<x> = PowerSeriesRing(GF(5))
sage: (1 + x + O(x^2)).exp()
Traceback (most recent call last):
...
ArithmeticError: constant term of power series does not support exponentiation
```

**exponents()**

Return the exponents appearing in self with nonzero coefficients.

**EXAMPLES:**

```
sage: R.<t> = PowerSeriesRing(QQ)
sage: f = t + t^2 - 10/3*t^3
sage: f.exponents()
sage: [1, 2, 3]
```

**inverse()**

Return the inverse of self, i.e., $self^{-1}$.

**EXAMPLES:**
sage: R.<t> = PowerSeriesRing(QQ, sparse=True)
sage: t.inverse()
t^-1
sage: type(_)
<type 'sage.rings.laurent_series_ring_element.LaurentSeries'>
sage: (1-t).inverse()
1 + t + t^2 + t^3 + t^4 + t^5 + t^6 + t^7 + t^8 + ...

**is_dense()**

**EXAMPLES:**

```
sage: R.<t> = PowerSeriesRing(ZZ)
sage: t.is_dense()
True
sage: R.<t> = PowerSeriesRing(ZZ, sparse=True)
sage: t.is_dense()
False
```

**is_gen()**

Return True if this is the generator (the variable) of the power series ring.

**EXAMPLES:**

```
sage: R.<t> = QQ[]
sage: t.is_gen()
True
sage: (1 + 2*t).is_gen()
False
```

Note that this only returns True on the actual generator, not on something that happens to be equal to it.

```
sage: (1*t).is_gen()
False
sage: 1*t == t
True
```

**is_monomial()**

Return True if this element is a monomial. That is, if self is \(x^n\) for some non-negative integer \(n\).

**EXAMPLES:**

```
sage: k.<z> = PowerSeriesRing(QQ, 'z')
sage: z.is_monomial()
True
sage: k(1).is_monomial()
True
sage: (z+1).is_monomial()
False
sage: (z^2909).is_monomial()
True
sage: (3*z^2909).is_monomial()
False
```

**is_sparse()**

**EXAMPLES:**

```
sage: R.<t> = PowerSeriesRing(ZZ)
sage: t.is_sparse()
```

(continues on next page)
is_square()
Return True if this function has a square root in this ring, e.g., there is an element \( y \) in \( \text{self.parent()} \) such that \( y^2 \) equals \( \text{self} \).

ALGORITHM: If the base ring is a field, this is true whenever the power series has even valuation and the leading coefficient is a perfect square.

For an integer domain, it attempts the square root in the fraction field and tests whether or not the result lies in the original ring.

EXAMPLES:

```python
sage: K.<t> = PowerSeriesRing(QQ, 't', 5)
sage: (1+t).is_square()
True
sage: (2+t).is_square()
False
sage: (2+t.change_ring(RR)).is_square()
True
sage: t.is_square()
False
sage: K.<t> = PowerSeriesRing(ZZ, 't', 5)
sage: (1+t).is_square()
False
sage: f = (1+t)^100
sage: f.is_square()
True
```

is_unit()
Return True if this power series is invertible.

A power series is invertible precisely when the constant term is invertible.

EXAMPLES:

```python
sage: R.<t> = PowerSeriesRing(ZZ)
sage: (-1 + t - t^5).is_unit()
True
sage: (3 + t - t^5).is_unit()
False
```

AUTHORS:
• David Harvey (2006-09-03)

laurent_series()
Return the Laurent series associated to this power series, i.e., this series considered as a Laurent series.

EXAMPLES:

```python
sage: k.<w> = QQ[[w]]
sage: f = 1+17*w+15*w^3+O(w^5)
sage: parent(f)
Power Series Ring in w over Rational Field
```
sage: g = f.laurent_series(); g
1 + 17*w + 15*w^3 + O(w^5)

lift_to_precision(absprec=None)

Return a congruent power series with absolute precision at least absprec.

INPUT:

- absprec – an integer or None (default: None), the absolute precision of the result. If None, lifts to an exact element.

EXAMPLES:

sage: A.<t> = PowerSeriesRing(GF(5))
sage: x = t + t^2 + O(t^5)
sage: x.lift_to_precision(10)
t + t^2 + O(t^10)
sage: x.lift_to_precision()
t + t^2

list()

See this method in derived classes:

- sage.rings.power_series_poly.PowerSeries_poly.list()
- sage.rings.multi_power_series_ring_element.MPowerSeries.list()

Implementations MUST override this in the derived class.

EXAMPLES:

sage: R.<x> = PowerSeriesRing(ZZ)
sage: PowerSeries.list(1+x^2)
Traceback (most recent call last):
  ... NotImplementedError

log(prec=None)

Return log of this power series to the indicated precision.

This works only if the constant term of the power series is 1 or the base ring can take the logarithm of the constant coefficient.

INPUT:

- prec – integer; default is self.parent().default_prec()

ALGORITHM: See solve_linear_de().

Warning: Screwy things can happen if the coefficient ring is not a field of characteristic zero. See solve_linear_de().

EXAMPLES:

sage: R.<t> = PowerSeriesRing(QQ, default_prec=10)
sage: (1 + t + O(t^10)).log()
t - 1/2*t^2 + 1/3*t^3 - 1/4*t^4 + 1/5*t^5 - 1/6*t^6 + 1/7*t^7 - 1/8*t^8 + 1/
-9*t^9 + O(t^10)
\begin{verbatim}
sage: t.exp().log()
t + O(t^10)
sage: (1+t).log().exp()
1 + t + O(t^10)
sage: (-1 + t + O(t^10)).log()
Traceback (most recent call last):
  ... ArithmeticError: constant term of power series is not 1

sage: R.<t> = PowerSeriesRing(RR)
sage: (2+t).log().exp()
2.00000000000000 + 1.00000000000000*t + O(t^20)
\end{verbatim}

\textbf{nth\_root} \((n, \text{prec=None})\)

Return the \(n\)-th root of this power series.

**INPUT:**

- \(n\) – integer
- \(\text{prec}\) – integer (optional) - precision of the result. Though, if this series has finite precision, then the result can not have larger precision.

**EXAMPLES:**

\begin{verbatim}
sage: R.<x> = QQ
sage: (1+x).nth_root(5)
1 + 1/5*x - 2/25*x^2 + ... + 12039376311816/2384185791015625*x^19 + O(x^20)
sage: (1 + x + O(x^5)).nth_root(5)
1 + 1/5*x - 2/25*x^2 + 6/125*x^3 - 21/625*x^4 + O(x^5)
\end{verbatim}

Check that the results are consistent with taking log and exponential:

\begin{verbatim}
sage: R.<x> = PowerSeriesRing(QQ, default_prec=100)
sage: p = (1 + 2*x - x^4)**200
sage: p1 = p.nth_root(1000, prec=100)
sage: p2 = (p.log()/1000).exp()
sage: p1.prec() == p2.prec() == 100
True
sage: p1.polynomial() == p2.polynomial()
True
\end{verbatim}

Positive characteristic:

\begin{verbatim}
sage: R.<u> = GF(3)
sage: p = 1 + 2*u^2
sage: p.nth_root(4)
1 + 2*u^2 + u^6 + 2*u^8 + u^12 + 2*u^14 + O(u^20)
sage: p.nth_root(4)**4
1 + 2*u^2 + O(u^20)
\end{verbatim}

\textbf{ogf\_to\_egf}() 

Returns the exponential generating function power series, assuming self is an ordinary generating function power series.
This can also be computed as \texttt{serconvol(f, exp(t))} in PARI/GP.

## Examples:

```
sage: R.<t> = PowerSeriesRing(QQ)
sage: f = t + t^2 + 2*t^3
sage: f.ogf_to_egf()
t + 1/2*t^2 + 1/3*t^3
```

### padded_list \(\texttt{(n=None)}\)

Return a list of coefficients of self up to (but not including) \(q^n\).
Includes 0’s in the list on the right so that the list has length \(n\).

**Input:**

- \(n\) - (optional) an integer that is at least 0. If \(n\) is not given, it will be taken to be the precision of self, unless this is +Infinity, in which case we just return \texttt{self.list()}.  

**Examples:**

```
sage: R.<q> = PowerSeriesRing(QQ)
sage: f = 1 - 17*q + 13*q^2 + 10*q^4 + O(q^7)
sage: f.list()
[1, -17, 13, 0, 10]
sage: f.padded_list(7)
[1, -17, 13, 0, 10, 0, 0]
sage: f.padded_list(10)
[1, -17, 13, 0, 10, 0, 0, 0, 0, 0]
sage: f.padded_list(3)
[1, -17, 13]
sage: f.padded_list()  
[1, -17, 13, 0, 10, 0, 0]
sage: g = 1 - 17*q + 13*q^2 + 10*q^4
sage: g.list()
[1, -17, 13, 0, 10]
sage: g.padded_list()  
[1, -17, 13, 0, 10]
sage: g.padded_list(10)
[1, -17, 13, 0, 10, 0, 0, 0, 0, 0]
```

### polynomial()

See this method in derived classes:

- \texttt{sage.rings.power_series_poly.PowerSeries_poly.polynomial()}.
- \texttt{sage.rings.multi_power_series_ring_element.MPowerSeries.polynomial()}

Implementations \textit{MUST} override this in the derived class.

**Examples:**

```
sage: R.<x> = PowerSeriesRing(ZZ)
sage: PowerSeries.polynomial(1+x^2)
Traceback (most recent call last):
  ...
NotImplementedError
```

### prec()

The precision of \(\ldots + O(x^r)\) is by definition \(r\).
EXAMPLES:

```python
sage: R.<t> = ZZ[[t]]
sage: (t^2 + O(t^3)).prec()
3
sage: (1 - t^2 + O(t^100)).prec()
100
```

**precision_absolute()**

Return the absolute precision of this series.

By definition, the absolute precision of \( \ldots + O(x^r) \) is \( r \).

EXAMPLES:

```python
sage: R.<t> = ZZ[[t]]
sage: (t^2 + O(t^3)).precision_absolute()
3
sage: (1 - t^2 + O(t^100)).precision_absolute()
100
```

**precision_relative()**

Return the relative precision of this series, that is the difference between its absolute precision and its valuation.

By convention, the relative precision of 0 (or \( O(x^r) \) for any \( r \)) is 0.

EXAMPLES:

```python
sage: R.<t> = ZZ[[t]]
sage: (t^2 + O(t^3)).precision_relative()
1
sage: (1 - t^2 + O(t^100)).precision_relative()
100
sage: 0(t^4).precision_relative()
0
```

**shift(n)**

Return this power series multiplied by the power \( t^n \). If \( n \) is negative, terms below \( t^n \) will be discarded. Does not change this power series.

**Note:** Despite the fact that higher order terms are printed to the right in a power series, right shifting decreases the powers of \( t \), while left shifting increases them. This is to be consistent with polynomials, integers, etc.

EXAMPLES:

```python
sage: R.<t> = PowerSeriesRing(QQ['y'], 't', 5)
sage: f = ~(1+t); f
1 - t + t^2 - t^3 + t^4 + O(t^5)
sage: f.shift(3)
t^3 - t^4 + t^5 - t^6 + t^7 + O(t^8)
sage: f >> 2
1 - t + t^2 + O(t^3)
sage: f << 10
t^10 - t^11 + t^12 - t^13 + t^14 + O(t^15)
sage: t << 29
t^30
```
AUTHORS:

- Robert Bradshaw (2007-04-18)

\( \text{sin}(\text{prec='infinity'}) \)

Apply \text{sin} to the formal power series.

INPUT:

- \text{prec} – Integer or \text{infinity}. The degree to truncate the result to.

OUTPUT:

A new power series.

EXAMPLES:

For one variable:

```
sage: t = PowerSeriesRing(QQ, 't').gen()
sage: f = (t + t**2).O(4)
sage: sin(f)
t + t^2 - 1/6*t^3 + O(t^4)
```

For several variables:

```
sage: T.<a,b> = PowerSeriesRing(ZZ,2)
sage: f = a + b + a*b + T.O(3)
sage: sin(f)
a + b + a*b + O(a, b)^3
sage: f.sin()
a + b + a*b + O(a, b)^3
sage: f.sin(prec=2)
a + b + O(a, b)^2
```

If the power series has a non-zero constant coefficient \(c\), one raises an error:

```
sage: g = 2+f
sage: sin(g)
Traceback (most recent call last):
... 
ValueError: Can only apply sin to formal power series with zero constant term.
```

If no precision is specified, the default precision is used:

```
sage: T.default_prec()
12
sage: sin(a)
a - 1/6*a^3 + 1/120*a^5 - 1/5040*a^7 + 1/362880*a^9 - 1/39916800*a^11 + O(a, b)^12
sage: a.sin(prec=5)
a - 1/6*a^3 + O(a, b)^5
sage: sin(a + T.O(5))
a - 1/6*a^3 + O(a, b)^5
```

\text{solve_linear_de}(\text{prec='infinity'}, b=None, f0=None)

Obtain a power series solution to an inhomogeneous linear differential equation of the form:

\[ f'(t) = a(t)f(t) + b(t). \]
• self - the power series $a(t)$
• b - the power series $b(t)$ (default is zero)
• $f_0$ - the constant term of $f$ (“initial condition”) (default is 1)
• prec - desired precision of result (this will be reduced if either a or b have less precision available)

OUTPUT: the power series $f$, to indicated precision

ALGORITHM: A divide-and-conquer strategy; see the source code. Running time is approximately $M(n) \log n$, where $M(n)$ is the time required for a polynomial multiplication of length $n$ over the coefficient ring. (If you’re working over something like $\mathbb{Q}$, running time analysis can be a little complicated because the coefficients tend to explode.)

Note:

• If the coefficient ring is a field of characteristic zero, then the solution will exist and is unique.
• For other coefficient rings, things are more complicated. A solution may not exist, and if it does it may not be unique. Generally, by the time the nth term has been computed, the algorithm will have attempted divisions by $n!$ in the coefficient ring. So if your coefficient ring has enough ‘precision’, and if your coefficient ring can perform divisions even when the answer is not unique, and if you know in advance that a solution exists, then this function will find a solution (otherwise it will probably crash).

AUTHORS:

• David Harvey (2006-09-11): factored functionality out from exp() function, cleaned up precision tests a bit

EXAMPLES:

```python
sage: R.<t> = PowerSeriesRing(QQ, default_prec=10)

sage: a = 2 - 3*t + 4*t^2 + O(t^10)
sage: b = 3 - 4*t^2 + O(t^7)

sage: f = a.solve_linear_de(prec=5, b=b, f0=3/5)
sage: f
3/5 + 21/5*t + 33/10*t^2 - 38/15*t^3 + 11/24*t^4 + O(t^5)

sage: f.derivative() - a*f - b
O(t^4)
```

```python
sage: a = 2 - 3*t + 4*t^2
sage: b = 3 - 4*t^2
sage: f = a.solve_linear_de(b=b, f0=3/5)
Traceback (most recent call last):
...
ValueError: cannot solve differential equation to infinite precision
```

```python
sage: a.solve_linear_de(prec=5, b=b, f0=3/5)
3/5 + 21/5*t + 33/10*t^2 - 38/15*t^3 + 11/24*t^4 + O(t^5)
```

$sqrt$ (prec=None, extend=False, all=False, name=None)

Return a square root of self.

INPUT:

• prec - integer (default: None): if not None and the series has infinite precision, truncates series at precision prec.
• extend - bool (default: False); if True, return a square root in an extension ring, if necessary. Otherwise, raise a ValueError if the square root is not in the base power series ring. For example, if extend is True the square root of a power series with odd degree leading coefficient is defined as an element of a formal extension ring.

• name - string; if extend is True, you must also specify the print name of the formal square root.

• all - bool (default: False); if True, return all square roots of self, instead of just one.

ALGORITHM: Newton’s method

\[ x_{i+1} = \frac{1}{2}(x_i + \text{self}/x_i) \]

EXAMPLES:

```python
sage: K.<t> = PowerSeriesRing(QQ, 't', 5)
sage: sqrt(t^2)
t
sage: sqrt(1+t)
1 + 1/2*t - 1/8*t^2 + 1/16*t^3 - 5/128*t^4 + O(t^5)
sage: sqrt(4*t)
2 + 1/4*t - 1/64*t^2 + 1/512*t^3 - 5/16384*t^4 + O(t^5)
sage: u = sqrt(2+t, prec=2, extend=True, name = 'alpha'); u
alpha
sage: u^2
2 + t
sage: u.parent()
Univariate Quotient Polynomial Ring in alpha over Power Series Ring in t over Rational Field with modulus x^2 - 2 - t
```

Univariate Quotient Polynomial Ring in alpha over Power Series Ring in t over Rational Field with modulus x^2 - 2 - t

```python
sage: K.<t> = PowerSeriesRing(QQ, 't', 50)
sage: sqrt(1+2*t+t^2)
t
sage: sqrt(t^2 +2*t^4 + t^6)
t + t^3
```

```python
sage: sqrt(1 + t + t^2 + 7*t^3)^2
1 + t + t^2 + 7*t^3 + O(t^50)
sage: sqrt(K(0))
0
sage: sqrt(t^2)
t
```

```python
sage: K.<t> = PowerSeriesRing(CDF, 5)
sage: v = sqrt(-1 + t + t^3, all=True); v
[1.0*I - 0.5*I*t - 0.125*I*t^2 - 0.5625*I*t^3 - 0.2890625*I*t^4 + O(t^5), -1.0*I + 0.5*I*t + 0.125*I*t^2 + 0.5625*I*t^3 + 0.2890625*I*t^4 + O(t^5)]
sage: [a^2 for a in v]
[-1.0 + 1.0*t + 0.0*t^2 + 1.0*t^3 + O(t^5), -1.0 + 1.0*t + 0.0*t^2 + 1.0*t^3 + O(t^5)]
```

A formal square root:

```python
sage: K.<t> = PowerSeriesRing(QQ, 5)
sage: f = 2*t + t^3 + O(t^4)
sage: s = f.sqrt(extend=True, name='sqrtf'); s
sqrtf
sage: s^2
2*t + t^3 + O(t^4)
sage: parent(s)
Univariate Quotient Polynomial Ring in sqrtf over Power Series Ring in t over Rational Field with modulus x^2 - 2*t - t^3 + O(t^4)
```
AUTHORS:

- Robert Bradshaw
- William Stein

\texttt{square\_root()} \\
Return the square root of self in this ring. If this cannot be done then an error will be raised.

This function succeeds if and only if \texttt{self.is\_square()}

EXAMPLES:

```python
sage: K.<t> = PowerSeriesRing(QQ, 't', 5)
sage: (1+t).square_root()
1 + 1/2*t - 1/8*t^2 + 1/16*t^3 - 5/128*t^4 + O(t^5)
sage: (2+t).square_root()
Traceback (most recent call last):
  ... ValueError: Square root does not live in this ring.
sage: (2+t.change_ring(RR)).square_root()
1.41421356237309 + 0.353553390593274*t - 0.0441941738241592*t^2 + 0.
  → 0.0110485434560398*t^3 - 0.00345266983001244*t^4 + O(t^5)
sage: t.square_root()
Traceback (most recent call last):
  ... ValueError: Square root not defined for power series of odd valuation.
sage: K.<t> = PowerSeriesRing(ZZ, 't', 5)
sage: f = (1+t)^20
sage: f.square_root()
1 + 10*t + 45*t^2 + 120*t^3 + 210*t^4 + O(t^5)
sage: f = 1+t
sage: f.square_root()
Traceback (most recent call last):
  ... ValueError: Square root does not live in this ring.
```

AUTHORS:

- Robert Bradshaw

\texttt{truncate}(\texttt{prec}='\texttt{infinity}') \\
The polynomial obtained from power series by truncation.

EXAMPLES:

```python
sage: R.<I> = GF(2)[[]]
sage: f = 1/(1+I+O(I^8)); f
1 + I + I^2 + I^3 + I^4 + I^5 + I^6 + I^7 + O(I^8)
sage: f.truncate(5)
I^4 + I^3 + I^2 + I + 1
```

\texttt{valuation}() \\
Return the valuation of this power series.

This is equal to the valuation of the underlying polynomial.

EXAMPLES:
Sparse examples:

```python
sage: R.<t> = PowerSeriesRing(QQ, sparse=True)
sage: f = t^100000 + O(t^10000000)
sage: f.valuation()
100000
sage: R(0).valuation()
+Infinity
```

Dense examples:

```python
sage: R.<t> = PowerSeriesRing(ZZ)
sage: f = 17*t^100 +O(t^110)
sage: f.valuation()
100
sage: t.valuation()
1
```

`valuation_zero_part()`

Factor self as \( q^n (a_0 + a_1 q + \cdots) \) with \( a_0 \) nonzero. Then this function returns \( a_0 + a_1 q + \cdots \).

**Note:** This valuation zero part need not be a unit if, e.g., \( a_0 \) is not invertible in the base ring.

**EXAMPLES:**

```python
sage: R.<t> = PowerSeriesRing(QQ)
sage: ((1/3)*t^5*(17-2/3*t^3)).valuation_zero_part()
17/3 - 2/9*t^3
```

In this example the valuation 0 part is not a unit:

```python
sage: R.<t> = PowerSeriesRing(ZZ, sparse=True)
sage: u = (-2*t^5*(17-t^3)).valuation_zero_part(); u
-34 + 2*t^3
sage: u.is_unit()
False
sage: u.valuation()
0
```

`variable()`

Return a string with the name of the variable of this power series.

**EXAMPLES:**

```python
sage: R.<x> = PowerSeriesRing(Rationals())
sage: f = x^2 + 3*x^4 + O(x^7)
sage: f.variable()
'x'
```

**AUTHORS:**

- David Harvey (2006-08-08)

`sage.rings.power_series_ring_element.is_PowerSeries(x)`

Return True if \( x \) is an instance of a univariate or multivariate power series.

**EXAMPLES:**
```
sage: R.<x> = PowerSeriesRing(ZZ)
sage: from sage.rings.power_series_ring_element import is_PowerSeries
sage: is_PowerSeries(1+x^2)
True
sage: is_PowerSeries(x-x)
True
sage: is_PowerSeries(0)
False
sage: var('x')
x
sage: is_PowerSeries(1+x^2)
False
```

```
sage.rings.power_series_ring_element.make_element_from_parent_v0(parent, *args)
sage.rings.power_series_ring_element.make_powerseries_poly_v0(parent, f, prec, is_gen)
```
The class `sage.rings.power_series_poly.PowerSeries_poly` provides additional methods for univariate power series.

```python
sage: R.<q> = PowerSeriesRing(CC)
sage: R
Power Series Ring in q over Complex Field with 53 bits of precision
sage: loads(q.dumps()) == q
True
sage: R.<t> = QQ
sage: f = 3 - t^3 + O(t^5)
sage: a = f^3; a
27 - 27*t^3 + O(t^5)
sage: b = f^-3; b
1/27 + 1/27*t^3 + O(t^5)
sage: a*b
1 + O(t^5)
```

Check that trac ticket #22216 is fixed:

```python
sage: R.<T> = PowerSeriesRing(QQ)
sage: R(pari('1 + O(T)'))
1 + O(T)
sage: R(pari('1/T + O(T)'))
Traceback (most recent call last):
  ... ValueError: series has negative valuation
```

### degree()

Return the degree of the underlying polynomial of self. That is, if self is of the form f(x) + O(x^n), we return the degree of f(x). Note that if f(x) is 0, we return -1, just as with polynomials.

```python
sage: R.<t> = ZZ
sage: (5 + t^3 + O(t^4)).degree()
3
sage: (5 + O(t^4)).degree()
0
sage: O(t^4).degree()
-1
```
dict()
Return a dictionary of coefficients for self. This is simply a dict for the underlying polynomial, so need not have keys corresponding to every number smaller than self.prec().

EXAMPLES:

```
sage: R.<t> = ZZ[[t]]
sage: f = 1 + t^10 + O(t^12)
sage: f.dict()
{0: 1, 10: 1}
```

integral(\text{\texttt{\textup{\textsc{var}=\texttt{\textend{verbatim}}}}})
The integral of this power series
By default, the integration variable is the variable of the power series.
Otherwise, the integration variable is the optional parameter \text{\texttt{\textup{\textsc{var}}}}

\textbf{Note:} The integral is always chosen so the constant term is 0.

EXAMPLES:

```
sage: k.<w> = QQ[[w]]
sage: (1+17*w+15*w^3+O(w^5)).integral()  
w + 17/2*w^2 + 15/4*w^4 + O(w^6)
sage: (w^3 + 4*w^4 + O(w^7)).integral()  
1/4*w^4 + 4/5*w^5 + O(w^8)
sage: (3*w^2).integral()  
w^3
```

list()
Return the list of known coefficients for self. This is just the list of coefficients of the underlying polynomial, so in particular, need not have length equal to self.prec().

EXAMPLES:

```
sage: R.<t> = ZZ[[t]]
sage: f = 1 - 5*t^3 + t^5 + O(t^7)
sage: f.list()
[1, 0, 0, -5, 0, 1]
```

pade(\text{\texttt{\textup{\textsc{m, n}}}})
Returns the Padé approximant of \text{\texttt{\textup{\textsc{self}}}} of index (\text{\texttt{\textsc{m, n}}}).

The Padé approximant of index (\text{\texttt{\textup{\textsc{m, n}}}}) of a formal power series \text{\texttt{\textup{\textsc{f}}}} is the quotient \text{\texttt{\textsc{Q}/P}} of two polynomials \text{\texttt{\textsc{Q}}} and \text{\texttt{\textsc{P}}} such that \text{\texttt{\textup{\textsc{deg(Q)}} \leq \text{\texttt{\textsc{m}}, \text{\texttt{deg(P)}} \leq \text{\texttt{n}}}}}
and

\[
f(z) - Q(z)/P(z) = O(z^{m+n+1}).\]

The formal power series \text{\texttt{\textup{\textsc{f}}}} must be known up to order \text{\texttt{\textup{\textsc{n+m+1}}}}.

See Wikipedia article Padé_approximant

INPUT:

\begin{itemize}
\item m, n – integers, describing the degrees of the polynomials
\end{itemize}

OUTPUT:

a ratio of two polynomials
Warning: The current implementation uses a very slow algorithm and is not suitable for high orders.

ALGORITHM:
This method uses the formula as a quotient of two determinants.

See also:
- `sage.matrix.berlekamp_massey`
- `sage.rings.polynomial.polynomial_zmod_flint.Polynomial_zmod_flint.rational_reconstruct`

EXAMPLES:
```
sage: z = PowerSeriesRing(QQ, 'z').gen()
sage: exp(z).pade(4, 0)
1/24*z^4 + 1/6*z^3 + 1/2*z^2 + z + 1
sage: exp(z).pade(1, 1)
(-z - 2)/(z - 2)
sage: exp(z).pade(3, 3)
(-z^3 - 12*z^2 - 60*z - 120)/(z^3 - 12*z^2 + 60*z - 120)
sage: log(1-z).pade(4, 4)
(25/6*z^4 - 130/3*z^3 + 105*z^2 - 70*z)/(z^4 - 20*z^3 + 90*z^2 - 140*z + 70)
sage: sqrt(1+z).pade(3, 2)
(1/6*z^3 + 3*z^2 + 8*z + 16/3)/(z^2 + 16/3*z + 16/3)
sage: exp(2*z).pade(3, 3)
(-z^3 - 6*z^2 - 15*z - 15)/(z^3 - 6*z^2 + 15*z - 15)
```
sage: R.<x> = PowerSeriesRing(QQ)
sage: f = 2*x + 3*x^2 - x^4 + O(x^5)
sage: g = f.reverse()
sage: g
1/2*x - 3/8*x^2 + 9/16*x^3 - 131/128*x^4 + O(x^5)
sage: f(g)
x + O(x^5)
sage: g(f)
x + O(x^5)

sage: A.<t> = PowerSeriesRing(ZZ)
sage: a = t - t^2 - 2*t^4 + t^5 + O(t^6)
sage: b = a.reverse(); b
t + t^2 + 2*t^3 + 7*t^4 + 25*t^5 + O(t^6)
sage: a(b)
t + O(t^6)
sage: b(a)
t + O(t^6)

sage: B.<b,c> = PolynomialRing(ZZ)
sage: A.<t> = PowerSeriesRing(B)
sage: f = t + b*t^2 + c*t^3 + O(t^4)
sage: g = f.reverse(); g
t - b*t^2 + (2*b^2 - c)*t^3 + O(t^4)
sage: f(g)
t + O(t^4)
sage: g(f)
t + O(t^4)

sage: A.<t> = PowerSeriesRing(ZZ)
sage: B.<s> = A[[[]]
sage: f = (1 - 3*t + 4*t^3 + O(t^4))*s + (2 + t + t^2 + O(t^3))*s^2 + O(s^3)
sage: set_verbose(1)
sage: g = f.reverse(); g
verbose 1 (<module>) passing to pari failed; trying Lagrange inversion
(1 + 3*t + 9*t^2 + 23*t^3 + O(t^4))*s + (-2 - 19*t - 118*t^2 + O(t^3))*s^2 + O(s^3)
˓→O(s^3)
sage: set_verbose(0)
sage: f(g) == g(f) == s
True

If the leading coefficient is not a unit, we pass to its fraction field if possible:
We can handle some base rings of positive characteristic:

```
sage: A8.<t> = PowerSeriesRing(Zmod(8))
sage: a = t - 15*t^2 - 2*t^4 + t^5 + O(t^6)
sage: b = a.reverse(); b
    t + 7*t^2 + 2*t^3 + 5*t^4 + t^5 + O(t^6)
sage: a(b)
    t + O(t^6)
sage: b(a)
    t + O(t^6)
```

The optional argument `precision` sets the precision of the output:

```
sage: R.<x> = PowerSeriesRing(QQ)
sage: f = 2*x + 3*x^2 - 7*x^3 + x^4 + O(x^5)
sage: g = f.reverse(precision=3); g
    1/2*x - 3/8*x^2 + O(x^3)
sage: f(g)
    x + O(x^3)
sage: g(f)
    x + O(x^3)
```

If the input series has infinite precision, the precision of the output is automatically set to the default precision of the parent ring:

```
sage: R.<I> = GF(2)<[[]]
sage: f = 1/(1+I+O(I^8)); f
    1 + I + I^2 + I^3 + I^4 + I^5 + I^6 + I^7 + O(I^8)
sage: f.truncate(5)
    1 + I + I^2 + I^3 + I^4 + O(I^5)
```

```
truncation (prec='infinity')
The polynomial obtained from power series by truncation at precision `prec`.

EXAMPLES:
```
sage: R.<I> = GF(2)[[]]
sage: f = 1/(1+I+O(I^8)); f
    1 + I + I^2 + I^3 + I^4 + I^5 + I^6 + I^7 + O(I^8)
sage: f.truncate_powerseries(5)
    I^4 + I^3 + I^2 + I + 1
```

```
valuation()
Return the valuation of self.

EXAMPLES:
`````


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

sage: (5 - t^8 + O(t^11)).valuation()
0
sage: (-t^8 + O(t^11)).valuation()
8
sage: O(t^7).valuation()
7
sage: R(0).valuation()
+Infinity
```

`sage.rings.power_series_poly.make_powerseries_poly_v0(parent, f, prec, is_gen)`

Return the power series specified by f, prec, and is_gen.

This function exists for the purposes of pickling. Do not delete this function – if you change the internal representation, instead make a new function and make sure that both kinds of objects correctly unpickle as the new type.

EXAMPLES:

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

sage: sage.rings.power_series_poly.make_powerseries_poly_v0(R, t, infinity, True)
```

36 Chapter 3. Power Series Methods
POWER SERIES IMPLEMENTED USING PARI

EXAMPLES:

This implementation can be selected for any base ring supported by PARI by passing the keyword implementation='pari' to the PowerSeriesRing() constructor:

```sage
default_k.<q> = PowerSeriesRing(ZZ, implementation='pari'); R
Power Series Ring in q over Integer Ring
default_k.<q> = PowerSeriesRing(CC, implementation='pari'); S
Power Series Ring in t over Complex Field with 53 bits of precision
```

Note that only the type of the elements depends on the implementation, not the type of the parents:

```sage
default_k.<q> = PowerSeriesRing(ZZ, implementation='pari'); R
default_k.<q> = PowerSeriesRing(CC, implementation='pari'); S
default_k.<q> = PowerSeriesRing(integers(), implementation='pari'); T
```

If $k$ is a finite field implemented using PARI, this is the default implementation for power series over $k$:

```sage
default_k.<c> = GF(5^12)
default_k.<c> = PowerSeriesRing(integers(), implementation='pari'); A
```

**Warning:** Because this implementation uses the PARI interface, the PARI variable ordering must be respected in the sense that the variable name of the power series ring must have higher priority than any variable names occurring in the base ring:

```sage
default_k.<q> = PowerSeriesRing(integers(), implementation='pari'); Q
```

Reversing the variable ordering leads to errors:

```sage
default_k.<x> = QQ[]
default_k.<x> = PowerSeriesRing(integers(), implementation='pari'); S
Traceback (most recent call last):
  ... PariError: incorrect priority in gtopoly: variable x <= y
```
AUTHORS:

- Peter Bruin (December 2013): initial version

```python
class sage.rings.power_series_pari.PowerSeries_pari
    Bases: sage.rings.power_series_ring_element.PowerSeries

A power series implemented using PARI.

INPUT:

- parent – the power series ring to use as the parent
- f – object from which to construct a power series
- prec – (default: infinity) precision of the element to be constructed
- check – ignored, but accepted for compatibility with PowerSeries_poly
```

dict()

Return a dictionary of coefficients for self.

This is simply a dict for the underlying polynomial; it need not have keys corresponding to every number smaller than self.prec().

EXAMPLES:

```python
sage: R.<t> = PowerSeriesRing(ZZ, implementation='pari')
sage: f = 1 + t^10 + O(t^12)
sage: f.dict()
{0: 1, 10: 1}
```

integral(var=None)

Return the formal integral of self.

By default, the integration variable is the variable of the power series. Otherwise, the integration variable is the optional parameter var.

Note: The integral is always chosen so the constant term is 0.

EXAMPLES:

```python
sage: k.<w> = PowerSeriesRing(QQ, implementation='pari')
sage: (1+17*w+15*w^3+O(w^5)).integral()
w + 17/2*w^2 + 15/4*w^4 + O(w^6)
sage: (w^3 + 4*w^4 + O(w^7)).integral()
1/4*w^4 + 4/5*w^5 + O(w^8)
sage: (3*w^2).integral()
w^3
```

list()

Return the list of known coefficients for self.

This is just the list of coefficients of the underlying polynomial; it need not have length equal to self.prec().

EXAMPLES:
sage: R.<t> = PowerSeriesRing(ZZ, implementation='pari')
sage: f = 1 - 5*t^3 + t^5 + O(t^7)
sage: f.list()
[1, 0, 0, -5, 0, 1]
sage: S.<u> = PowerSeriesRing(pAdicRing(5), implementation='pari')
sage: (2 + u).list()
[2 + O(5^20), 1 + O(5^20)]

padded_list (n=None)
Return a list of coefficients of self up to (but not including) $q^n$.
The list is padded with zeroes on the right so that it has length $n$.

INPUT:

- n -- a non-negative integer (optional); if n is not given, it will be taken to be the precision of self, unless this is ``+Infinity, in which case we just return self.list()

EXAMPLES:

sage: R.<q> = PowerSeriesRing(QQ, implementation='pari')
sage: f = 1 - 17*q + 13*q^2 + 10*q^4 + O(q^7)
sage: f.list()
[1, -17, 13, 0, 10]
sage: f.padded_list(7)
[1, -17, 13, 0, 10, 0, 0]
sage: f.padded_list(10)
[1, -17, 13, 0, 10, 0, 0, 0, 0, 0]
sage: f.padded_list(3)
[1, -17, 13]
sage: f.padded_list()
[1, -17, 13, 0, 10, 0, 0]
sage: g = 1 - 17*q + 13*q^2 + 10*q^4
sage: g.list()
[1, -17, 13, 0, 10]
sage: g.padded_list()
[1, -17, 13, 0, 10]
sage: g.padded_list(10)
[1, -17, 13, 0, 10, 0, 0, 0, 0, 0]

polynomial()
Convert self to a polynomial.

EXAMPLES:

sage: R.<t> = PowerSeriesRing(GF(7), implementation='pari')
sage: f = 3 - t^3 + O(t^5)
sage: f.polynomial()
6*t^3 + 3

reverse (precision=None)
Return the reverse of self.
The reverse of a power series $f$ is the power series $g$ such that $g(f(x)) = x$. This exists if and only if the valuation of self is exactly 1 and the coefficient of x is a unit.

If the optional argument precision is given, the reverse is returned with this precision. If f has infinite precision and the argument precision is not given, then the reverse is returned with the default precision of f.parent().
EXAMPLES:

```python
sage: R.<x> = PowerSeriesRing(QQ, implementation='pari')
sage: f = 2*x + 3*x^2 - x^4 + O(x^5)
sage: g = f.reverse()
sage: g
definition of g
sage: f(g)
x + O(x^5)
sage: g(f)
x + O(x^5)

sage: A.<t> = PowerSeriesRing(ZZ, implementation='pari')
sage: a = t - 2*t^2 - 2*t^4 + t^5 + O(t^6)
sage: b = a.reverse(); b
definition of b
t + t^2 + 2*t^3 + 7*t^4 + 25*t^5 + O(t^6)
sage: a(b)
t + O(t^6)
sage: b(a)
t + O(t^6)

sage: B.<b,c> = PolynomialRing(ZZ)
sage: A.<t> = PowerSeriesRing(B, implementation='pari')
sage: f = t + b*t^2 + c*t^3 + O(t^4)
sage: g = f.reverse(); g
definition of g
t - b*t^2 + (2*b^2 - c)*t^3 + O(t^4)
sage: f(g)
t + O(t^4)
sage: g(f)
t + O(t^4)

sage: A.<t> = PowerSeriesRing(QQ, implementation='pari')
sage: B.<x> = PowerSeriesRing(A, implementation='pari')
sage: f = (1 - 3*t + 4*t^3 + O(t^4))*x + (2 + t + t^2 + O(t^3))*x^2 + O(x^3)
sage: g = f.reverse(); g

continues on next page

The optional argument `precision` sets the precision of the output:

```python
sage: R.<x> = PowerSeriesRing(QQ, implementation='pari')
sage: f = 2*x + 3*x^2 - x^4 + O(x^5)
sage: g = f.reverse(precision=3); g
definition of g
sage: f(g)
x + O(x^3)
sage: g(f)
x + O(x^3)

If the input series has infinite precision, the precision of the output is automatically set to the default precision of the parent ring:

```python
sage: R.<x> = PowerSeriesRing(QQ, default_prec=20, implementation='pari')
sage: f = f.reverse()  

continues on next page

(continues on next page)
### valuation()

Return the valuation of `self`.

**EXAMPLES:**

```python
sage: R.<t> = PowerSeriesRing(QQ, implementation='pari')
sage: (5 - t^8 + O(t^11)).valuation()
0
sage: (-t^8 + O(t^11)).valuation()
8
sage: O(t^7).valuation()
7
sage: R(0).valuation()
+Infinity
```
Construct a multivariate power series ring (in finitely many variables) over a given (commutative) base ring.

EXAMPLES:

Construct rings and elements:

```
sage: R.<t,u,v> = PowerSeriesRing(QQ); R
Multivariate Power Series Ring in t, u, v over Rational Field
sage: TestSuite(R).run()
sage: p = -t + 1/2*t^3*u - 1/4*t^4*u + 2/3*v^5 + R.O(6); p
-t + 1/2*t^3*u - 1/4*t^4*u + 2/3*v^5 + O(t, u, v)^6
sage: p in R
True
sage: g = 1 + v + 3*t^2*u - 2*t^2*v^2; g
1 + v + 3*t^2*u - 2*t^2*v^2
sage: g in R
True
```

Add big O as with single variable power series:

```
sage: g.add_bigoh(3)
1 + v + O(t, u, v)^3
sage: g = g.O(5); g
1 + v + 3*t^2*u - 2*t^2*v^2 + O(t, u, v)^5
```

Sage keeps track of total-degree precision:

```
sage: f = (g-1)^2 - g + 1; f
-v + v^2 - 3*t^2*u + 6*t^2*u*v + 2*t^2*v^2 + O(t, u, v)^5
sage: f in R
True
sage: f.prec()
5
sage: ((g-1-v)^2).prec()
8
```

Construct multivariate power series rings over various base rings.

```
sage: M = PowerSeriesRing(QQ, 4, 'k'); M
Multivariate Power Series Ring in k0, k1, k2, k3 over Rational Field
sage: loads(dumps(M)) is M
True
sage: TestSuite(M).run()
```

(continues on next page)
sage: H = PowerSeriesRing(PolynomialRing(ZZ,3,'z'),4,'f'); H
Multivariate Power Series Ring in f0, f1, f2, f3 over Multivariate Polynomial Ring in z0, z1, z2 over Integer Ring
sage: TestSuite(H).run()
sage: loads(dumps(H)) is H
True
sage: z = H.base_ring().gens()
sage: f = H.gens()
+ H.O(3)
sage: h in H
True
sage: h
4*z1^2 + 2*z0*z2 + z1*z2 + z2^2 + (-z2^2 - 2*z0 + z2)*f0*f2 \
+ (-22*z0^2 + 2*z1^2 - z0*z2 + z2^2 - 1955*z2)*f1*f2 \
+ (2*z0*z1 + z[1]*z[2] - z[2]^2 - z[1] + 3*z[2])*f3^2 \
+ O(f0, f1, f2, f3)^3
sage: S.<x,y> = PowerSeriesRing(GF(65537)); S
Multivariate Power Series Ring in x, y over Finite Field of size 65537
sage: s = -30077*x + 9485*x*y - 6260*y^3 + 12870*x^2*y^2 - 20289*y^4 + S.O(5); s
-30077*x + 9485*x*y - 6260*y^3 + 12870*x^2*y^2 - 20289*y^4 + O(x, y)^5
sage: s in S
True
sage: TestSuite(S).run()
sage: loads(dumps(S)) is S
True

• Use angle-bracket notation:

sage: ZZ[['s,t,u']]
Multivariate Power Series Ring in s, t, u over Integer Ring
sage: GF(127931)[['x,y']]
Multivariate Power Series Ring in x, y over Finite Field of size 127931

Variable ordering determines how series are displayed.

sage: T.<a,b> = PowerSeriesRing(ZZ,order='deglex'); T
Multivariate Power Series Ring in a, b over Integer Ring
sage: TestSuite(T).run()
sage: loads(dumps(T)) is T
True
sage: T.term_order()
Degree lexicographic term order
sage: p = - 2*b^6 + a^5*b^2 + a^7 - b^2 - a*b^3 + T.O(9); p
a^7 + a^5*b^2 - 2*b^6 - a*b^3 - b^2 + O(a, b)^9
sage: U = PowerSeriesRing(ZZ,'a,b',order='negdeglex'); U
Multivariate Power Series Ring in a, b over Integer Ring

(continues on next page)
sage: U.term_order()
Negative degree lexicographic term order
sage: U(p)
-b^2 - a*b^3 - 2*b^6 + a^7 + a^5*b^2 + O(a, b)^9

Change from one base ring to another:

sage: R.<t,u,v> = PowerSeriesRing(QQ); R
Multivariate Power Series Ring in t, u, v over Rational Field
sage: R.base_extend(RR)
Multivariate Power Series Ring in t, u, v over Real Field with 53 bits of precision
sage: R.change_ring(IntegerModRing(10))
Multivariate Power Series Ring in t, u, v over Ring of integers modulo 10
sage: S = PowerSeriesRing(GF(65537),2,'x,y'); S
Multivariate Power Series Ring in x, y over Finite Field of size 65537
sage: S.change_ring(GF(5))
Multivariate Power Series Ring in x, y over Finite Field of size 5

Coercion from polynomial ring:

sage: R.<t,u,v> = PowerSeriesRing(QQ); R
Multivariate Power Series Ring in t, u, v over Rational Field
sage: A = PolynomialRing(QQ,3,'t,u,v')

sage: g = A.gens()

sage: a = 2*g[0]*g[2] - 2*g[0] - 2; a
2*t*v - 2*t - 2
sage: R(a)
-2 - 2*t + 2*t*v
sage: R(a).O(4)
-2 - 2*t + 2*t*v + O(t, u, v)^4
sage: a.parent()
Multivariate Polynomial Ring in t, u, v over Integer Ring
sage: a in R
True

Coercion from polynomial ring in subset of variables:

sage: R.<t,u,v> = PowerSeriesRing(QQ); R
Multivariate Power Series Ring in t, u, v over Rational Field
sage: A = PolynomialRing(QQ,2,'t,v')

sage: g = A.gens()

sage: a = -2*g[0]*g[1] - 1/27*g[1]^2 + g[0] - 1/2*g[1]; a
-2*t*v - 1/27*v^2 + t - 1/2*v
sage: a in R
True

Coercion from symbolic ring:

sage: x,y = var('x,y')
sage: S = PowerSeriesRing(GF(11),2,'x,y'); S
Multivariate Power Series Ring in x, y over Finite Field of size 11
sage: type(x)
<type 'sage.symbolic.expression.Expression'>
The implementation of the multivariate power series ring uses a combination of multivariate polynomials and univariate power series. Namely, in order to construct the multivariate power series ring \( R[[x_1, x_2, \cdots, x_n]] \), we consider the univariate power series ring \( S[[T]] \) over the multivariate polynomial ring \( S := R[x_1, x_2, \cdots, x_n] \), and in it we take the subring formed by all power series whose \( i \)-th coefficient has degree \( i \) for all \( i \geq 0 \). This subring is isomorphic to \( R[[x_1, x_2, \cdots, x_n]] \). This is how \( R[[x_1, x_2, \cdots, x_n]] \) is implemented in this class. The ring \( S \) is called the foreground polynomial ring, and the ring \( S[[T]] \) is called the background univariate power series ring.

AUTHORS:

- Niles Johnson (2010-07): initial code
- Simon King (2012-08, 2013-02): Use category and coercion framework, trac ticket #13412 and trac ticket #14084

class sage.rings.multi_power_series_ring.MPowerSeriesRing_generic(base_ring, num_gens, name_list, order='negdeglex', default_prec=10, sparse=False)


A multivariate power series ring. This class is implemented as a single variable power series ring in the variable \( T \) over a multivariable polynomial ring in the specified generators. Each generator \( g \) of the multivariable polynomial ring (called the “foreground ring”) is mapped to \( g \cdot T \) in the single variable power series ring (called the “background ring”). The background power series ring is used to do arithmetic and track total-degree precision. The foreground polynomial ring is used to display elements.

For usage and examples, see above, and PowerSeriesRing().

Element

Element alias of sage.rings.multi_power_series_ring_element.MPowerSeries

O (prec)

Return big oh with precision \( \text{prec} \). This function is an alias for bigoh.

EXAMPLES:

```
sage: T.<a,b> = PowerSeriesRing(ZZ,2); T
Multivariate Power Series Ring in a, b over Integer Ring
sage: T.O(10)
0 + O(a, b)^10
```

Chapter 5. Multivariate Power Series Rings
bigoh\((\text{prec})\)
Return big oh with precision \(\text{prec}\). The function \(O\) does the same thing.

EXAMPLES:

```python
sage: T.<a,b> = PowerSeriesRing(ZZ,2); T
Multivariate Power Series Ring in a, b over Integer Ring
sage: T.bigoh(10)
0 + O(a, b)^10
sage: T.O(10)
0 + O(a, b)^10
```

change_ring\((R)\)
Returns the power series ring over \(R\) in the same variable as self. This function ignores the question of whether the base ring of self is or can extend to the base ring of \(R\); for the latter, use base_extend.

EXAMPLES:

```python
sage: R.<t,u,v> = PowerSeriesRing(QQ); R
Multivariate Power Series Ring in t, u, v over Rational Field
sage: R.base_extend(RR)
Multivariate Power Series Ring in t, u, v over Real Field with
53 bits of precision
sage: R.change_ring(IntegerModRing(10))
Multivariate Power Series Ring in t, u, v over Ring of integers
modulo 10
sage: R.base_extend(IntegerModRing(10))
Traceback (most recent call last):
...
TypeError: no base extension defined
```

characteristic()
Return characteristic of base ring, which is characteristic of self.

EXAMPLES:

```python
sage: H = PowerSeriesRing(GF(65537),4,'f'); H
Multivariate Power Series Ring in f0, f1, f2, f3 over
Finite Field of size 65537
sage: H.characteristic()
65537
```

construction()
Returns a functor \(F\) and base ring \(R\) such that \(F(R) == \text{self}\).

EXAMPLES:

```python
sage: M = PowerSeriesRing(QQ,4,'f'); M
Multivariate Power Series Ring in f0, f1, f2, f3 over Rational Field
sage: (c,R) = M.construction(); (c,R)
(Completion[('f0', 'f1', 'f2', 'f3'), prec=12],
```
Multivariate Polynomial Ring in f0, f1, f2, f3 over Rational Field
sage: c = Completion([('f0', 'f1', 'f2', 'f3'), prec=12]

sage: c(R)
Multivariate Power Series Ring in f0, f1, f2, f3 over Rational Field
sage: c(R) == M
True

gen(n=0)
Return the nth generator of self.

EXAMPLES:

sage: M = PowerSeriesRing(ZZ,10,'v')
sage: M.gen(6)
v6

is_dense()
Is self dense? (opposite of sparse)

EXAMPLES:

sage: M = PowerSeriesRing(ZZ,3,'s,t,u'); M
Multivariate Power Series Ring in s, t, u over Integer Ring
sage: M.is_dense()
True
sage: N = PowerSeriesRing(ZZ,3,'s,t,u',sparse=True); N
Sparse Multivariate Power Series Ring in s, t, u over Integer Ring
sage: N.is_dense()
False

is_integral_domain(proof=False)
Return True if the base ring is an integral domain; otherwise return False.

EXAMPLES:

sage: M = PowerSeriesRing(QQ,4,'v'); M
Multivariate Power Series Ring in v0, v1, v2, v3 over Rational Field
sage: M.is_integral_domain()
True

is_noetherian(proof=False)
Power series over a Noetherian ring are Noetherian.

EXAMPLES:

sage: M = PowerSeriesRing(QQ,4,'v'); M
Multivariate Power Series Ring in v0, v1, v2, v3 over Rational Field
sage: M.is_noetherian()
True

sage: W = PowerSeriesRing(InfinitePolynomialRing(ZZ,'a'),2,'x,y')
sage: W.is_noetherian()
False

is_sparse()
Is self sparse?

EXAMPLES:
```python
sage: M = PowerSeriesRing(ZZ,3,'s,t,u'); M
Multivariate Power Series Ring in s, t, u over Integer Ring
sage: M.is_sparse()
False
sage: N = PowerSeriesRing(ZZ,3,'s,t,u',sparse=True); N
Sparse Multivariate Power Series Ring in s, t, u over Integer Ring
sage: N.is_sparse()
True
```

```python
laurent_series_ring()
Laurent series not yet implemented for multivariate power series rings
```

```python
ngens()
Return number of generators of self.

EXAMPLES:
```
```python
sage: M = PowerSeriesRing(ZZ,10,'v')
sage: M.ngens()
10
```

```python
prec_ideal()
Return the ideal which determines precision; this is the ideal generated by all of the generators of our background polynomial ring.

EXAMPLES:
```python
sage: A.<s,t,u> = PowerSeriesRing(ZZ)
sage: A.prec_ideal()
Ideal (s, t, u) of Multivariate Polynomial Ring in s, t, u over Integer Ring
```

```python
remove_var(*var)
Remove given variable or sequence of variables from self.

EXAMPLES:
```
```python
sage: A.<s,t,u> = PowerSeriesRing(ZZ)
sage: A.remove_var(t)
Multivariate Power Series Ring in s, u over Integer Ring
sage: A.remove_var(s,t)
Power Series Ring in u over Integer Ring
sage: M = PowerSeriesRing(GF(5),5,'t'); M
Multivariate Power Series Ring in t0, t1, t2, t3, t4 over Finite Field of size 5
sage: M.remove_var(M.gens()[3])
Multivariate Power Series Ring in t0, t1, t2, t4 over Finite Field of size 5
```

Removing all variables results in the base ring:
```
```python
sage: M.remove_var(*M.gens())
Finite Field of size 5
```

```python
term_order()
Print term ordering of self. Term orderings are implemented by the TermOrder class.
```
EXAMPLES:

```python
sage: M.<x,y,z> = PowerSeriesRing(ZZ,3)
sage: M.term_order()
Negative degree lexicographic term order
sage: m = y*z^12 - y^6*z^8 - x^7*y^5*z^2 + x*y^2*z + M.O(15); m
x*y^2*z + y*z^12 - x^7*y^5*z^2 - y^6*z^8 + O(x, y, z)^15
sage: N = PowerSeriesRing(ZZ,3,'x,y,z', order="deglex")
sage: N.term_order()
Degree lexicographic term order
sage: N(m)
-x^7*y^5*z^2 - y^6*z^8 + y*z^12 + x*y^2*z + O(x, y, z)^15
```

`sage.rings.multi_power_series_ring.is_MPowerSeriesRing(x)`

Return true if input is a multivariate power series ring.

`sage.rings.multi_power_series_ring.unpickle_multi_power_series_ring_v0(base_ring, num_gens, names, order, default_prec, sparse)`

Unpickle (deserialize) a multivariate power series ring according to the given inputs.

EXAMPLES:

```python
sage: P.<x,y> = PowerSeriesRing(QQ)
sage: loads(dumps(P)) == P  # indirect doctest
True
```
MULTIVARIATE POWER SERIES

Construct and manipulate multivariate power series (in finitely many variables) over a given commutative ring. Multivariate power series are implemented with total-degree precision.

EXAMPLES:

Power series arithmetic, tracking precision:

\[
\begin{align*}
\text{sage: } & R.<s,t> = \text{PowerSeriesRing}(\mathbb{Z}); R \\
& \text{Multivariate Power Series Ring in } s, t \text{ over Integer Ring} \\
\text{sage: } & f = 1 + s + 3s^2; f \\
& 1 + s + 3s^2 \\
\text{sage: } & g = t^2s + 3s^2t^2 + O(s, t)^5; g \\
& st^2 + 3s^2t^2 + 0(s, t)^5 \\
\text{sage: } & f = f.O(7); f \\
& 1 + s + 3s^2 + O(s, t)^7 \\
\text{sage: } & f += s; f \\
& 1 + 2s + 3s^2 + O(s, t)^7 \\
\text{sage: } & f*g \\
& st^2 + 5s^2t^2 + O(s, t)^5 \\
\text{sage: } & (f-1)*g \\
& 2s^2t^2 + 9s^3t^2 + O(s, t)^6 \\
\text{sage: } & f*g - g \\
& 2s^2t^2 + O(s, t)^5 \\
\text{sage: } & f*=s; f \\
& s + 2s^2 + 3s^3 + O(s, t)^8 \\
\text{sage: } & f%2 \\
& s + s^3 + O(s, t)^8 \\
\text{sage: } & (f%2).parent() \\
& \text{Multivariate Power Series Ring in } s, t \text{ over Ring of integers modulo 2}
\end{align*}
\]

As with univariate power series, comparison of \( f \) and \( g \) is done up to the minimum precision of \( f \) and \( g \):

\[
\begin{align*}
\text{sage: } & f = 1 + t + s + s*t + R.O(3); f \\
& 1 + s + t + s*t + 0(s, t)^3 \\
\text{sage: } & g = s^2 + 2s^4 - s^5 + s^2t^3 + R.O(6); g \\
& s^2 + 2s^4 - s^5 + s^2t^3 + 0(s, t)^6 \\
\text{sage: } & f == g \\
& \text{False} \\
\text{sage: } & g == g.add_bigoh(3) \\
& \text{True}
\end{align*}
\]

(continues on next page)
Calling:

\[
\begin{align*}
\text{sage: } & f = s^2 + s \cdot t + s^3 + s^2 \cdot t + 3 \cdot s^4 + 3 \cdot s^3 \cdot t + R.O(5); \ f \\
& s^2 + s \cdot t + s^3 + s^2 \cdot t + 3 \cdot s^4 + 3 \cdot s^3 \cdot t + O(s, t)^5 \\
\text{sage: } & f(t, s) \\
& s \cdot t + t^2 + s \cdot t^2 + t^3 + 3 \cdot s \cdot t^3 + 3 \cdot t^4 + O(s, t)^5 \\
\text{sage: } & f(t^2, s^2) \\
& s^2 \cdot t^2 + t^4 + s^2 \cdot t^4 + t^6 + 3 \cdot s^2 \cdot t^6 + 3 \cdot t^8 + O(s, t)^10
\end{align*}
\]

Substitution is defined only for elements of positive valuation, unless \( f \) has infinite precision:

\[
\begin{align*}
\text{sage: } & f(t^2, s^2 + 1) \\
\text{Traceback (most recent call last):} \\
... \\
\text{TypeError: Substitution defined only for elements of positive valuation, unless self has infinite precision.}
\end{align*}
\]

\[
\begin{align*}
\text{sage: } & g = f.truncate() \\
\text{sage: } & g(t^2, s^2 + 1) \\
& t^2 + s^2 \cdot t^2 + 2 \cdot t^4 + s^2 \cdot t^4 + 4 \cdot t^6 + 3 \cdot s^2 \cdot t^6 + 3 \cdot t^8 \\
\text{sage: } & g(t^2, (s^2 + 1).O(3)) \\
& t^2 + s^2 \cdot t^2 + 2 \cdot t^4 + O(s, t)^5
\end{align*}
\]

0 has valuation \(+\infty\):

\[
\begin{align*}
\text{sage: } & f(t^2, 0) \\
& t^4 + t^6 + 3 \cdot t^8 + O(s, t)^{10} \\
\text{sage: } & f(t^2, s^2 + s) \\
& s \cdot t^2 + s^2 \cdot t^2 + t^4 + O(s, t)^5
\end{align*}
\]

Substitution of power series with finite precision works too:

\[
\begin{align*}
\text{sage: } & f(s.O(2), t) \\
& s^2 + s \cdot t + O(s, t)^3 \\
\text{sage: } & f(f, f) \\
& 2 \cdot s^4 + 4 \cdot s^3 \cdot t + 2 \cdot s^2 \cdot t^2 + 2 \cdot s^4 \cdot t^4 + 4 \cdot s^5 \cdot t^4 + 4 \cdot s^4 \cdot t^6 + 2 \cdot s^3 \cdot t^6 + 16 \cdot s^6 + 34 \cdot s^5 \cdot t \\
& + 20 \cdot s^4 \cdot t^2 + 2 \cdot s^3 \cdot t^3 + O(s, t)^7 \\
\text{sage: } & t(f, f) \\
& s^2 + s \cdot t + s^3 + s^2 \cdot t + 3 \cdot s^4 + 3 \cdot s^3 \cdot t + O(s, t)^5 \\
\text{sage: } & t(0, f) == s(f, 0) \\
& True
\end{align*}
\]

The \texttt{subs} syntax works as expected:

\[
\begin{align*}
\text{sage: } & r0 = -t^2 - s \cdot t^3 - 2 \cdot t^6 + s^7 + s^5 \cdot t^2 + R.O(10) \\
\text{sage: } & r1 = s^4 - s^3 \cdot t + s^6 \cdot t - 4 \cdot s^2 \cdot t^5 - 6 \cdot s^3 \cdot t^5 + R.O(10) \\
\text{sage: } & r2 = 2 \cdot s^3 \cdot t^2 - 2 \cdot s^4 \cdot t - 2 \cdot s^3 \cdot t^4 + s^7 + t^2 + R.O(10) \\
\text{sage: } & r0.subs((t:r2, s:r1)) \\
& -4 \cdot s^6 \cdot t^4 + 8 \cdot s^4 \cdot t^6 - 4 \cdot s^2 \cdot t^8 + 8 \cdot s^6 \cdot t^6 - 8 \cdot s^4 \cdot t^8 - 4 \cdot s^4 \cdot t^9 \\
& + 4 \cdot s^2 \cdot t^{11} - 4 \cdot s^6 \cdot t^8 + O(s, t)^{15} \\
\text{sage: } & r0.subs((t:r2, s:r1)) == r0(r1, r2) \\
& True
\end{align*}
\]
Construct ring homomorphisms from one power series ring to another:

```
sage: A.<a,b> = PowerSeriesRing(QQ)
sage: X.<x,y> = PowerSeriesRing(QQ)
sage: phi = Hom(A,X)([x,2*y]); phi
Ring morphism:
  From: Multivariate Power Series Ring in a, b over Rational Field
  To:  Multivariate Power Series Ring in x, y over Rational Field
    Defn: a |---> x
         b |---> 2*y
sage: phi(a+b+3*a*b^2 + A.O(5))
x + 2*y + 12*x*y^2 + O(x, y)^5
```

Multiplicative inversion of power series:

```
sage: h = 1 + s + t + s*t + s^2*t^2 + 3*s^4 + 3*s^3*t + R.O(5)
sage: k = h^-1; k
1 - s - t + s^2 + s*t + t^2 - s^3 - s^2*t - t^3 - 2*s^4 -
  2*s^3*t + s*t^2 + t^4 + O(s, t)^5
sage: h*k
1 + O(s, t)^5
```

AUTHORS:

- Niles Johnson (07/2010): initial code
- Simon King (08/2012): Use category and coercion framework, trac ticket #13412
class sage.rings.multi_power_series_ring_element.MPowerSeries(parent, x=0, prec=+Infinity, is_gen=False, check=False)

Bases: sage.rings.power_series_ring_element.PowerSeries

Multivariate power series; these are the elements of Multivariate Power Series Rings.

INPUT:

• **parent** – A multivariate power series.

• **x** – The element (default: 0). This can be another *MPowerSeries* object, or an element of one of the following:
  – the background univariate power series ring
  – the foreground polynomial ring
  – a ring that coerces to one of the above two

• **prec** – (default: infinity) The precision

• **is_gen** – (default: False) Is this element one of the generators?

• **check** – (default: False) Needed by univariate power series class

EXAMPLES:

Construct multivariate power series from generators:

```
sage: S.<s,t> = PowerSeriesRing(ZZ)
sage: f = s + 4*s*t + 3*s^t
go in S
True
sage: f = f.add_bigoh(4); f
s + 4*s*t + 3*s^t + O(s, t)^4
sage: g = 1 + s*t + s^t + S.O(5); g
1 + s + t - s*t + S.O(5); g
1 + s + t - s*t + O(s, t)^5
```

```
sage: T = PowerSeriesRing(GF(3),5,'t'); T
Multivariate Power Series Ring in t0, t1, t2, t3, t4 over Finite
Field of size 3
sage: t = T.gens()
t0 + t1+t3 - t4^3 - t0^3*t2^2
sage: w = w.add_bigoh(5); w
t0 + t1+t3 - t4^3 + O(t0, t1, t2, t3, t4)^5
sage: w in T
True

```
Get random elements:

```
sage: S.random_element(4)  # random
-2*t + t^2 - 12*s^3 + O(s, t)^4
sage: T.random_element(10) # random
-t1^2*t3^2*t4^2 + t1^5*t3^3*t4 + O(t0, t1, t2, t3, t4)^10
```

Convert elements from polynomial rings:

```
sage: R = PolynomialRing(ZZ,5,T.variable_names())
sage: t = R.gens()
sage: T(r)
-t2*t3 + t3^2 + t4^2
sage: r.parent()
Multivariate Polynomial Ring in t0, t1, t2, t3, t4 over Integer Ring
sage: r in T
True
```

```
O(prec)
Return a multivariate power series of precision prec obtained by truncating self at precision prec.
This is the same as add_bigoh().
```

```
EXAMPLES:
```
```
sage: B.<x,y> = PowerSeriesRing(QQ); B
Multivariate Power Series Ring in x, y over Rational Field
sage: r = 1 - x*y + x^2
sage: r.O(4)
1 + x^2 - x*y + O(x, y)^4
sage: r.O(2)
1 + O(x, y)^2
```

```
Note that this does not change self:
```
```
sage: r
1 + x^2 - x*y
```

```
V(n)
If
```
```
f = \sum a_{m_0,...,m_k} x_0^{m_0} \cdots x_k^{m_k},
```
```
then this function returns
```
```
\sum a_{m_0,...,m_k} x_0^{nm_0} \cdots x_k^{nm_k}.
```
```
The total-degree precision of the output is n times the precision of self.
EXAMPLES:
```
```
sage: H = QQ[['x,y,z']]
sage: (x,y,z) = H.gens()
sage: h = -x*y^4*z^7 - 1/4*y*z^12 + 1/2*x^7*y^5*z^2 + 2/3*y^6*z^8 + H.O(15)
sage: h.V(3)
-x^3*y^12*z^21 - 1/4*y^3*z^36 + 1/2*x^21*y^15*z^6 + 2/3*y^18*z^24 + O(x, y, z)^45

add_bigoh(prec)
Return a multivariate power series of precision prec obtained by truncating self at precision prec.

EXAMPLES:

sage: B.<x,y> = PowerSeriesRing(QQ); B
Multivariate Power Series Ring in x, y over Rational Field
sage: r = 1 - x*y + x^2
sage: r.add_bigoh(4)
1 + x^2 - x*y + O(x, y)^4
sage: r.add_bigoh(2)
1 + O(x, y)^2

Note that this does not change self:

sage: r
1 + x^2 - x*y

coefficients()
Return a dict of monomials and coefficients.

EXAMPLES:

sage: R.<s,t> = PowerSeriesRing(ZZ); R
Multivariate Power Series Ring in s, t over Integer Ring
sage: f = 1 + t + s + s*t + R.O(3)
sage: f.coefficients()
{1: 1, s: 1, t: 1, s*t: 1}
sage: (f^2).coefficients()
{t^2: 1, 2*s*t: 1, s^2: 1, t: 2, s: 2, 1: 1}
sage: g = f^2 + f - 2; g
3*s + 3*t + s^2 + 5*s*t + t^2 + O(s, t)^3
sage: cd = g.coefficients()
sage: g2 = sum(k*v for (k,v) in cd.items()); g2
3*s + 3*t + s^2 + 5*s*t + t^2
sage: g2 == g.truncate()
True

constant_coefficient()
Return constant coefficient of self.

EXAMPLES:

sage: R.<a,b,c> = PowerSeriesRing(ZZ); R
Multivariate Power Series Ring in a, b, c over Integer Ring
sage: f = 3 + a + b - a*b - b*c - a*c + R.O(4)
sage: f.constant_coefficient()
sage: f.constant_coefficient().parent()
Integer Ring

\textbf{degree()}

Return degree of underlying polynomial of self.

\textbf{EXAMPLES:}

\begin{verbatim}
sage: B.<x,y> = PowerSeriesRing(QQ)
sage: B
Multivariate Power Series Ring in x, y over Rational Field
sage: r = 1 - x*y + x^2
sage: r = r.add_bigoh(4); r
1 + x^2 - x*y + O(x, y)^4
sage: r.de"
\texttt{egf}()

Method from univariate power series not yet implemented

\texttt{exp}(\texttt{prec=}+\texttt{Infinity})

Exponentiate the formal power series.

INPUT:

\begin{itemize}
  \item \texttt{prec} – Integer or \texttt{infinity}. The degree to truncate the result to.
\end{itemize}

OUTPUT:

The exponentiated multivariate power series as a new multivariate power series.

EXAMPLES:

\begin{verbatim}
sage: T.<a,b> = PowerSeriesRing(ZZ,2)
sage: f = a + b + a*b + T.O(3)
sage: exp(f)
 1 + a + b + 1/2*a^2 + 2*a*b + 1/2*b^2 + O(a, b)^3
sage: f.exp()
 1 + a + b + 1/2*a^2 + 2*a*b + 1/2*b^2 + O(a, b)^3
sage: f.exp(prec=2)
 1 + a + b + O(a, b)^2
sage: log(exp(f)) - f
 0 + O(a, b)^3
\end{verbatim}

If the power series has a constant coefficient $c$ and \texttt{exp}(c) is transcendental, then \texttt{exp}(f) would have to be a power series over the \texttt{SymbolicRing}. These are not yet implemented and therefore such cases raise an error:

\begin{verbatim}
sage: g = 2+f
sage: exp(g)
Traceback (most recent call last):
  ...
TypeError: unsupported operand parent(s) for *: 'Symbolic Ring' and 'Power Series Ring in Tbg over Multivariate Polynomial Ring in a, b over Rational Field'
\end{verbatim}

Another workaround for this limitation is to change base ring to one which is closed under exponentiation, such as \texttt{R} or \texttt{C}:

\begin{verbatim}
sage: exp(g.change_ring(RDF))
7.38905609... + 7.38905609...*a + 7.38905609...*b + 3.69452804...*a^2 + 14.7781121...*a*b + 3.69452804...*b^2 + O(a, b)^3
\end{verbatim}

If no precision is specified, the default precision is used:

\begin{verbatim}
sage: T.default_prec()
12
sage: exp(a)
 1 + a + 1/2*a^2 + 1/6*a^3 + 1/24*a^4 + 1/120*a^5 + 1/720*a^6 + 1/5040*a^7 + 1/40320*a^8 + 1/362880*a^9 + 1/3628800*a^10 + 1/39916800*a^11 + O(a, b)^12
sage: a.exp(prec=5)
 1 + a + 1/2*a^2 + 1/6*a^3 + 1/24*a^4 + O(a, b)^5
sage: exp(a + T.O(5))
 1 + a + 1/2*a^2 + 1/6*a^3 + 1/24*a^4 + O(a, b)^5
\end{verbatim}

\texttt{exponents}()

Return a list of tuples which hold the exponents of each monomial of \texttt{self}.
EXAMPLES:

```python
sage: H = QQ[\['x,y'\)]
sage: (x,y) = H.gens()
sage: h = -y^2 - x*y^3 - 6/5*y^6 - x^7 + 2*x^5*y^2 + H.O(10)
sage: h
-y^2 - x*y^3 - 6/5*y^6 - x^7 + 2*x^5*y^2 + O(x, y)^10
sage: h.exponents()
[(0, 2), (1, 3), (0, 6), (7, 0), (5, 2)]
```

```
integral(*args)
The formal integral of this multivariate power series, with respect to variables supplied in args.
The variable sequence args can contain both variables and counts; for the syntax, see derivative_parse().

EXAMPLES:

```python
sage: T.<a,b> = PowerSeriesRing(QQ,2)
sage: f = a + b + a^2*b + T.O(5)
sage: f.integral(a, 2)
1/6*a^3 + 1/2*a^2*b + 1/12*a^4*b + O(a, b)^7
sage: f.integral(a, b)
1/2*a^2*b + 1/2*a*b^2 + 1/6*a^3*b^2 + O(a, b)^7
sage: f.integral(a, 5)
1/720*a^6 + 1/120*a^5*b + 1/2520*a^7*b + O(a, b)^10
```

Only integration with respect to variables works:

```python
sage: f.integral(a+b)
Traceback (most recent call last):
...  
ValueError: a + b is not a variable
```

**Warning:** Coefficient division.

If the base ring is not a field (e.g. \( \mathbb{Z} \)), or if it has a non-zero characteristic, (e.g. \( \mathbb{Z}/3\mathbb{Z} \)), integration is not always possible while staying with the same base ring. In the first case, Sage will report that it has not been able to coerce some coefficient to the base ring:

```python
sage: T.<a,b> = PowerSeriesRing(ZZ,2)
sage: f = a + T.O(5)
sage: f.integral(a)
Traceback (most recent call last):
...  
TypeError: no conversion of this rational to integer
```

One can get the correct result by changing the base ring first:

```python
sage: f.change_ring(QQ).integral(a)
1/2*a^2 + O(a, b)^6
```

However, a correct result is returned even without base change if the denominator cancels:

```python
sage: f = 2*b + T.O(5)
sage: f.integral(b)
b^2 + O(a, b)^6
```

In non-zero characteristic, Sage will report that a zero division occurred.
```sage
t.<a,b> = PowerSeriesRing(Zmod(3),2)
(a^3).integral(a)
a^4
(a^2).integral(a)
Traceback (most recent call last):
...
ZeroDivisionError: inverse of Mod(0, 3) does not exist
```

**is_nilpotent()**

Return **True** if *self* is nilpotent. This occurs if

- *self* has finite precision and positive valuation, or
- *self* is constant and nilpotent in base ring.

Otherwise, return **False**.

**Warning:** This is so far just a sufficient condition, so don’t trust a **False** output to be legit!

**Todo:** What should we do about this method? Is nilpotency of a power series even decidable (assuming a nilpotency oracle in the base ring)? And I am not sure that returning **True** just because the series has finite precision and zero constant term is a good idea.

**EXAMPLES:**

```sage
R.<a,b,c> = PowerSeriesRing(Zmod(8)); R
Multivariate Power Series Ring in a, b, c over Ring of integers modulo 8
f = a + b + c + a^2*c
f.is_nilpotent()
False
f = f.O(4); f
a + b + c + a^2*c + O(a, b, c)^4
f.is_nilpotent()
True
g = R(2)
g.is_nilpotent()
True
g.O(4).is_nilpotent()
True
S = R.change_ring(QQ)
S(g).is_nilpotent()
False
S(g.O(4)).is_nilpotent()
False
```

**is_square()**

Method from univariate power series not yet implemented.

**is_unit()**

A multivariate power series is a unit if and only if its constant coefficient is a unit.
EXAMPLES:

```
sage: R.<a,b> = PowerSeriesRing(ZZ); R
Multivariate Power Series Ring in a, b over Integer Ring
sage: f = 2 + a^2 + a*b + a^3 + R.O(9)
sage: f.is_unit()
False
sage: f.base_extend(QQ).is_unit()
True
```

```
laurent_series()
Not implemented for multivariate power series.
```

```
list()
Doesn’t make sense for multivariate power series. Multivariate polynomials don’t have list of coefficients either.
```

```
log(prec=+Infinity)
Return the logarithm of the formal power series.
```

**INPUT:**

- `prec` – Integer or infinity. The degree to truncate the result to.

**OUTPUT:**

The logarithm of the multivariate power series as a new multivariate power series.

**EXAMPLES:**

```
sage: T.<a,b> = PowerSeriesRing(ZZ,2)
sage: f = 1 + a + b + a*b + T.O(5)
sage: f.log()
a + b - 1/2*a^2 - 1/2*b^2 + 1/3*a^3 + 1/3*b^3 - 1/4*a^4 - 1/4*b^4 + O(a, b)^5
sage: log(f)
a + b - 1/2*a^2 - 1/2*b^2 + 1/3*a^3 + 1/3*b^3 - 1/4*a^4 - 1/4*b^4 + O(a, b)^5
sage: exp(log(f)) - f
0 + O(a, b)^5
```

If the power series has a constant coefficient $c$ and $\exp(c)$ is transcendental, then $\exp(f)$ would have to be a power series over the `SymbolicRing`. These are not yet implemented and therefore such cases raise an error:

```
sage: g = 2+f
sage: log(g)
Traceback (most recent call last):
  ... TypeError: unsupported operand parent(s) for -: 'Symbolic Ring' and 'Power Series Ring in Tbg over Multivariate Polynomial Ring in a, b over Rational → Field'
```

Another workaround for this limitation is to change base ring to one which is closed under exponentiation, such as $\mathbb{R}$ or $\mathbb{C}$:

```
sage: log(g.change_ring(RDF))
1.09861228... + 0.33333333...*a + 0.33333333...*b - 0.0555555555...*a^2 + 0.0222222222...*a*b - 0.0555555555...*b^2 + 0.0123456790...*a^3 - 0.0740740740...*a^2*b - 0.0740740740...*a*b^2 + 0.0123456790...*b^3 - 0.00308641975...*a^4 + 0.0246913580...*a^3*b + 0.0246913580...*a*b^3 - 0.00308641975...*b^4 + O(a, b)^5
```

```
61
```
monomials()  
Return a list of monomials of self.  
These are the keys of the dict returned by coefficients().  

EXAMPLES:

```
sage: R.<a,b,c> = PowerSeriesRing(ZZ); R  
Multivariate Power Series Ring in a, b, c over Integer Ring  
sage: f = 1 + a + b - a*b - b*c - a*c + R.O(4)  
sage: sorted(f.monomials())  
[b*c, a*c, a*b, b, a, 1]  
sage: f = 1 + 2*a + 7*b - 2*a*b - 4*b*c - 13*a*c + R.O(4)  
sage: sorted(f.monomials())  
[b*c, a*c, a*b, b, a, 1]  
sage: f = R.zero()  
sage: f.monomials()  
[]
```

ogf()  
Method from univariate power series not yet implemented

padded_list()  
Method from univariate power series not yet implemented.

polynomial()  
Return the underlying polynomial of self as an element of the underlying multivariate polynomial ring (the "foreground polynomial ring").  

EXAMPLES:

```
sage: M = PowerSeriesRing(QQ,4,'t'); M  
Multivariate Power Series Ring in t0, t1, t2, t3 over Rational Field  
sage: t = M.gens()  
sage: f  
1/2*t0^3*t1^3*t2^2 + 2/3*t0*t2^6*t3 - t0^3*t1^3*t3^3 - 1/4*t0*t1*t2^7 + O(t0, t1, t2, t3)^10  
sage: f.polynomial()  
1/2*t0^3*t1^3*t2^2 + 2/3*t0*t2^6*t3 - t0^3*t1^3*t3^3 - 1/4*t0*t1*t2^7  
sage: f.polynomial().parent()  
Multivariate Polynomial Ring in t0, t1, t2, t3 over Rational Field
```

Contrast with truncate():

```
sage: f.truncate()  
1/2*t0^3*t1^3*t2^2 + 2/3*t0*t2^6*t3 - t0^3*t1^3*t3^3 - 1/4*t0*t1*t2^7  
sage: f.truncate().parent()  
Multivariate Power Series Ring in t0, t1, t2, t3 over Rational Field
```

prec()  
Return precision of self.  

EXAMPLES:
```python
sage: R.<a,b,c> = PowerSeriesRing(ZZ); R
Multivariate Power Series Ring in a, b, c over Integer Ring
sage: f = 3 + a + b - a*b - b*c - a*c + R.O(4)
sage: f.prec()
4
sage: f.truncate().prec()
+Infinity
```

```
quo_rem(other, precision=None)

Return the pair of quotient and remainder for the increasing power division of self by other.

If \(a \) and \(b\) are two elements of a power series ring \(R[[x_1, x_2, \cdots, x_n]]\) such that the trailing term of \(b\) is invertible in \(R\), then the pair of quotient and remainder for the increasing power division of \(a\) by \(b\) is the unique pair \((u, v) \in R[[x_1, x_2, \cdots, x_n]] \times R[x_1, x_2, \cdots, x_n]\) such that \(a = bu + v\) and such that no monomial appearing in \(v\) divides the trailing monomial (\(\text{trailing\_monomial()}\)) of \(b\). Note that this depends on the order of the variables.

This method returns both quotient and remainder as power series, even though in mathematics, the remainder for the increasing power division of two power series is a polynomial. This is because Sage’s power series come with a precision, and that precision is not always sufficient to determine the remainder completely. Disregarding this issue, the \(\text{polynomial()}\) method can be used to recast the remainder as an actual polynomial.

INPUT:

- \(\text{other} - \) an element of the same power series ring as \(\text{self}\) such that the trailing term of \(\text{other}\) is invertible in \(\text{self}\) (this is automatically satisfied if the base ring is a field, unless \(\text{other}\) is zero)
- \(\text{precision} - \) (default: the default precision of the parent of \(\text{self}\)) nonnegative integer, determining the precision to be cast on the resulting quotient and remainder if both \(\text{self}\) and \(\text{other}\) have infinite precision (ignored otherwise); note that the resulting precision might be lower than this integer

EXAMPLES:

```python
sage: R.<a,b,c> = PowerSeriesRing(ZZ)
sage: f = 1 + a + b - a*b + R.O(3)
sage: g = 1 + 2*a - 3*a*b + R.O(3)

sage: q, r = f.quo_rem(g); q, r
(1 - a + b + 2*a^2 + O(a, b, c)^3, 0 + O(a, b, c)^3)
sage: f == q*g+r
True

sage: q, r = (a*f).quo_rem(g); q, r
(a - a^2 + a*b + 2*a^3 + O(a, b, c)^4, 0 + O(a, b, c)^4)
sage: a*f == q*(a*g)+r
True

sage: q, r = (a*f).quo_rem(a*g); q, r
(1 - a + b + 2*a^2 + O(a, b, c)^3, 0 + O(a, b, c)^4)
sage: a*f == q*(a*g)+r
True

sage: q, r = (a*f).quo_rem(b*g); q, r
(a - 3*a^2 + O(a, b, c)^3, a + a^2 + O(a, b, c)^4)
sage: a*f == q*(b*g)+r
True
```

Trying to divide two polynomials, we run into the issue that there is no natural setting for the precision of the quotient and remainder (and if we wouldn’t set a precision, the algorithm would never terminate).
Here, default precision comes to our help:

```
sage: (1+a^3).quo_rem(a+a^2)
(1, a^2 + a^3 - a^5 + a^6 - a^7 + a^8 - a^9 + O(a, b, c)^11, l + O(a, b, c)^12)

default precision:
sage: (1+a^3+a*b).quo_rem(b+c)
(a + O(a, b, c)^11, l - a*c + a^3 + O(a, b, c)^12)
```

```
sage: (1+a^3+a*b).quo_rem(b+c, precision=17)
(a + O(a, b, c)^16, l - a*c + a^3 + O(a, b, c)^17)
```

```
sage: (1+a^3+a*b).quo_rem(b+c)
(a - b + O(a, b, c)^11, 2*b^2 + 2*b*c + 2*c^2 + O(a, b, c)^12)
```

```
sage: (1+a^3+a*b).quo_rem(1/(1+a+b+c))
(a^2 + b^2 + c^2 + a^3 + a^2*b + a^2*c + a*b^2 + a*c^2 + b^3 + b^2*c + b*c^2, l + c^3 + O(a, b, c)^14)
```

Illustrating the dependency on the ordering of variables:

```
sage: (1+a+b).quo_rem(b+c)
(1 + O(a, b, c)^11, l + a - c + O(a, b, c)^12)
```

```
sage: (1+b+c).quo_rem(c+a)
(0 + O(a, b, c)^11, l + b + c + O(a, b, c)^12)
```

```
sage: (1+c+a).quo_rem(a+b)
(0 + O(a, b, c)^11, l - b + c + O(a, b, c)^12)
```

**shift**

Doesn’t make sense for multivariate power series.

**solve_linear_de** *(prec=+Infinity, b=None, f0=None)*

Not implemented for multivariate power series.

**sqrt**

Method from univariate power series not yet implemented. Depends on square root method for multivariate polynomials.

**square_root**

Method from univariate power series not yet implemented. Depends on square root method for multivariate polynomials.

**trailing_monomial**

Return the trailing monomial of self.

This is defined here as the lowest term of the underlying polynomial.

EXAMPLES:
.. code::

    sage: R.<a,b,c> = PowerSeriesRing(ZZ)
    sage: f = 1 + a + b - a*b + R.O(3)
    sage: f.trailing_monomial()
    1
    sage: f = a^2*b^3*f; f
    a^2*b^3 + a^3*b^3 + a^2*b^4 - a^3*b^4 + O(a, b, c)^8
    sage: f.trailing_monomial()
    a^2*b^3

.. code::

    sage: f = PowerSeriesRing(QQ,4,'t'); M
    Multivariate Power Series Ring in t0, t1, t2, t3 over Rational Field
    sage: t = M.gens()
    sage: f
    1/2*t0^3*t1^3*t2^2 + 2/3*t0*t2^6*t3 - t0^3*t1^3*t3^3 - 1/4*t0*t1*t2^7 + O(t0, t1, t2, t3)^10
    sage: f.truncate()
    1/2*t0^3*t1^3*t2^2 + 2/3*t0*t2^6*t3 - t0^3*t1^3*t3^3 - 1/4*t0*t1*t2^7
    sage: f.truncate().parent()
    Multivariate Power Series Ring in t0, t1, t2, t3 over Rational Field

Contrast with polynomial:

.. code::

    sage: f.polynomial()
    1/2*t0^3*t1^3*t2^2 + 2/3*t0*t2^6*t3 - t0^3*t1^3*t3^3 - 1/4*t0*t1*t2^7
    sage: f.polynomial().parent()
    Multivariate Polynomial Ring in t0, t1, t2, t3 over Rational Field

valuation()

Return the valuation of self.

The valuation of a power series \( f \) is the highest nonnegative integer \( k \) less or equal to the precision of \( f \) and such that the coefficient of \( f \) before each term of degree < \( k \) is zero. (If such an integer does not exist, then the valuation is the precision of \( f \) itself.)

EXAMPLES:

.. code::

    sage: R.<a,b> = PowerSeriesRing(GF(4949717)); R
    Multivariate Power Series Ring in a, b over Finite Field of size 4949717
    sage: f = a^2 + a*b + a^3 + R.O(9)
    sage: f.valuation()
    2
    sage: g = 1 + a + a^3
    sage: g.valuation()
    0
    sage: R.zero().valuation()
    +Infinity

valuation_zero_part()

Doesn’t make sense for multivariate power series; valuation zero with respect to which variable?
variable()
Doesn’t make sense for multivariate power series.

variables()
Return tuple of variables occurring in self.

EXAMPLES:

```
sage: T = PowerSeriesRing(GF(3),5,'t'); T
Multivariate Power Series Ring in t0, t1, t2, t3, t4 over
Finite Field of size 3
sage: t = T.gens()
sage: w
t0 + t0*t2 - t4^3 - t0^3*t2^2 + O(t0, t1, t2, t3, t4)^6
sage: w.variables()
(t0, t2, t4)
```

sage.rings.multi_power_series_ring_element.is_MPowerSeries(f)
Return True if f is a multivariate power series.
EXAMPLES:

```sage
sage: R = LaurentSeriesRing(QQ, "x")
sage: R.base_ring()
Rational Field
sage: S = LaurentSeriesRing(GF(17)['x', 'y'])
sage: S
Laurent Series Ring in y over Univariate Polynomial Ring in x over Finite Field of size 17
sage: S.base_ring()
Univariate Polynomial Ring in x over Finite Field of size 17
```

See also:

- `sage.misc.defaults.set_series_precision()`

```python
class sage.rings.laurent_series_ring.LaurentSeriesRing(power_series)
    Bases: sage.structure.unique_representation.UniqueRepresentation, sage.rings.ring.CommutativeRing

Univariate Laurent Series Ring.
```

EXAMPLES:

```sage
sage: R = LaurentSeriesRing(QQ, 'x'); R
Laurent Series Ring in x over Rational Field
sage: x = R.0
sage: g = 1 - x + x^2 - x^4 +O(x^8); g
1 - x + x^2 - x^4 + O(x^8)
sage: g = 10*x^-3 + 2006 - 19*x + x^2 - x^4 +O(x^8); g
10*x^-3 + 2006 - 19*x + x^2 - x^4 + O(x^8)
```

You can also use more mathematical notation when the base is a field:

```sage
sage: Frac(QQ[['x']])
Laurent Series Ring in x over Rational Field
sage: Frac(GF(5)['y'])
Fraction Field of Univariate Polynomial Ring in y over Finite Field of size 5
```

Here the fraction field is not just the Laurent series ring, so you can’t use the `Frac` notation to make the Laurent series ring:

```sage
sage: Frac(ZZ[['t']])
Fraction Field of Power Series Ring in t over Integer Ring
```

Laurent series rings are determined by their variable and the base ring, and are globally unique:
When the base ring $k$ is a field, the ring $k((x))$ is a CDVF, that is a field equipped with a discrete valuation for which it is complete. The appropriate (sub)category is automatically set in this case:

```python
sage: k = GF(11)
sage: R.<x> = k[[x]]
sage: F = Frac(R)
sage: F.category()
Category of infinite complete discrete valuation fields
sage: TestSuite(F).run()
```

### Element

alias of `sage.rings.laurent_series_ring_element.LaurentSeries`

### base_extend($R$)

Return the Laurent series ring over $R$ in the same variable as self, assuming there is a canonical coerce map from the base ring of self to $R$.

**EXAMPLES:**

```python
sage: K.<x> = LaurentSeriesRing(QQ, default_prec=4)
sage: K.base_extend(QQ['t'])
Laurent Series Ring in x over Univariate Polynomial Ring in t over Rational
```

### change_ring($R$)

**EXAMPLES:**

```python
sage: K.<x> = LaurentSeriesRing(QQ, default_prec=4)
sage: R = K.change_ring(ZZ); R
Laurent Series Ring in x over Integer Ring
sage: R.default_prec()
4
```

### characteristic()

**EXAMPLES:**
construction()

Return the functorial construction of this Laurent power series ring.

The construction is given as the completion of the Laurent polynomials.

EXAMPLES:

```sage
sage: L.<t> = LaurentSeriesRing(ZZ, default_prec=42)
sage: phi, arg = L.construction()
sage: phi
Completion[t, prec=42]
sage: arg
Univariate Laurent Polynomial Ring in t over Integer Ring
sage: phi(arg) is L
True
```

Because of this construction, pushout is automatically available:

```sage
sage: 1/2 * t
1/2*t
sage: parent(1/2 * t)
Laurent Series Ring in t over Rational Field
sage: QQbar.gen() * t
I*t
sage: parent(QQbar.gen() * t)
Laurent Series Ring in t over Algebraic Field
```

default_prec()

Get the precision to which exact elements are truncated when necessary (most frequently when inverting).

EXAMPLES:

```sage
sage: R.<x> = LaurentSeriesRing(QQ, default_prec=5)
sage: R.default_prec()
5
```

fraction_field()

Return the fraction field of this ring of Laurent series.

If the base ring is a field, then Laurent series are already a field. If the base ring is a domain, then the Laurent series over its fraction field is returned. Otherwise, raise a ValueError.

EXAMPLES:

```sage
sage: R = LaurentSeriesRing(ZZ, 't', 30).fraction_field()
sage: R
Laurent Series Ring in t over Rational Field
sage: R.default_prec()
30
sage: LaurentSeriesRing(Zmod(4), 't').fraction_field()
Traceback (most recent call last):
...
ValueError: must be an integral domain
```
The `gen` method:

```python
sage: R = LaurentSeriesRing(QQ, "x")
sage: R.gen()
```

The `is_dense` method:

```python
sage: K.<x> = LaurentSeriesRing(QQ, sparse=True)
sage: K.is_dense()
False
```

The `is_exact` method:

```python
sage: R = LaurentSeriesRing(QQ, "x")
sage: R.is_exact()
False
```

The `is_field` method:

```python
sage: R = LaurentSeriesRing(QQ, "x")
sage: R.is_field(proof=True)
```

The `is_sparse` method:

```python
sage: K.<x> = LaurentSeriesRing(QQ, sparse=True)
sage: K.is_sparse()
True
```

The `laurent_polynomial_ring` method:

```python
sage: R = LaurentSeriesRing(QQ, "x")
sage: R.laurent_polynomial_ring()
```

The `ngens` method:

```python
sage: R = LaurentSeriesRing(QQ, "x")
sage: R.ngens()
1
```

The `polynomial_ring` method:

```python
sage: R = LaurentSeriesRing(QQ, "x")
sage: R.polynomial_ring()
```

The `gen` method allows you to access the generator of the Laurent series ring. The `is_dense` method checks if the ring is dense, and `is_exact` checks if the ring is exact. The `is_field` method determines if the ring is a field (where the base ring is a field). The `is_sparse` method checks if the ring is sparse. The `laurent_polynomial_ring` method returns the Laurent polynomial ring, `polynomial_ring` returns the polynomial ring, and `ngens` returns the number of generators.
sage: R = LaurentSeriesRing(QQ, "x")

Univariate Polynomial Ring in x over Rational Field

power_series_ring()
If this is the Laurent series ring \( R((t)) \), return the power series ring \( R[[t]] \).

EXAMPLES:

sage: R = LaurentSeriesRing(QQ, "x")

sage: R.power_series_ring()

Power Series Ring in x over Rational Field

residue_field()
Return the residue field of this Laurent series field if it is a complete discrete valuation field (i.e. if the base ring is a field, in which base it is also the residue field).

EXAMPLES:

sage: R.<x> = LaurentSeriesRing(GF(17))

sage: R.residue_field()

Finite Field of size 17

sage: R.<x> = LaurentSeriesRing(ZZ)

sage: R.residue_field()

Traceback (most recent call last):
...
TypeError: the base ring is not a field

uniformizer()
Return a uniformizer of this Laurent series field if it is a discrete valuation field (i.e. if the base ring is actually a field). Otherwise, an error is raised.

EXAMPLES:

sage: R.<t> = LaurentSeriesRing(QQ)

sage: R.uniformizer()

Traceback (most recent call last):
...
TypeError: the base ring is not a field

sage.rings.laurent_series_ring.is_LaurentSeriesRing(x)
Return **True** if this is a **univariate** Laurent series ring.

This is in keeping with the behavior of **is_PolynomialRing** versus **is_MPolynomialRing**.
EXAMPLES:

```
sage: R.<t> = LaurentSeriesRing(GF(7), 't'); R
Laurent Series Ring in t over Finite Field of size 7
sage: f = 1/(1-t+O(t^10)); f
1 + t + t^2 + t^3 + t^4 + t^5 + t^6 + t^7 + t^8 + t^9 + O(t^10)
```

Laurent series are immutable:

```
sage: f[2]  
1
Traceback (most recent call last):
  ...  
IndexError: Laurent series are immutable
```

We compute with a Laurent series over the complex mpfr numbers.

```
sage: K.<q> = Frac(CC[['q']])
sage: K
Laurent Series Ring in q over Complex Field with 53 bits of precision
sage: q
1.00000000000000*q
```

Saving and loading.

```
sage: loads(q.dumps()) == q
True
sage: loads(K.dumps()) == K
True
```

IMPLEMENTATION: Laurent series in Sage are represented internally as a power of the variable times the unit part (which need not be a unit - it’s a polynomial with nonzero constant term). The zero Laurent series has unit part 0.

AUTHORS:

- William Stein: original version
- David Joyner (2006-01-22): added examples
- Robert Bradshaw: Cython version

```python
class sage.rings.laurent_series_ring_element.LaurentSeries
    Bases: sage.structure.element.AlgebraElement
```
A Laurent Series.
We consider a Laurent series of the form $t^n \cdot f$ where $f$ is a power series.

**INPUT:**
- `parent` – a Laurent series ring
- `f` – a power series (or something can be coerced to one); note that $f$ does not have to be a unit
- `n` – (default: 0) integer

**$O(prec)$**
Return the Laurent series of precision at most `prec` obtained by adding $O(q^{prec})$, where $q$ is the variable.

The precision of `self` and the integer `prec` can be arbitrary. The resulting Laurent series will have precision equal to the minimum of the precision of `self` and `prec`. The term $O(q^{prec})$ is the zero series with precision `prec`.

**EXAMPLES:**
```
sage: R.<t> = LaurentSeriesRing(QQ)
sage: f = t^-5 + t^-4 + t^3 + O(t^10); f  
t^-5 + t^-4 + t^3 + O(t^10)
sage: f.O(-4)  
t^-5 + O(t^-4)
sage: f.O(15)  
t^-5 + t^-4 + t^3 + O(t^10)
```

**add_bigoh**

**EXAMPLES:**
```
sage: R.<t> = LaurentSeriesRing(QQ)
sage: f = t^2 + t^3 + O(t^10); f  
t^2 + t^3 + O(t^10)
sage: f.add_bigoh(5)  
t^2 + t^3 + O(t^5)
```

**change_ring** ($R$)
Change the base ring of `self`.

**EXAMPLES:**
```
sage: R.<q> = LaurentSeriesRing(ZZ)
sage: p = R([1,2,3]); p  
1 + 2*q + 3*q^2
sage: p.change_ring(GF(2))  
1 + q^2
```

**coefficients**
Return the nonzero coefficients of `self`.

**EXAMPLES:**
```
sage: R.<t> = LaurentSeriesRing(QQ)
sage: f = -5/t^(2) + t + t^2 - 10/3*t^3  
[-5, 1, 1, -10/3]
```

**common_prec** ($other$)
Return the minimum precision of `self` and `other`. 
### EXAMPLES:

```
sage: R.<t> = LaurentSeriesRing(QQ)
sage: f = t^(-1) + t + t^2 + O(t^3)
sage: g = t + t^3 + t^4 + O(t^4)
sage: f.common_prec(g)
3
sage: g.common_prec(f)
3

sage: f = t + t^2 + O(t^3)
sage: g = t^(-3) + t^2
sage: f.common_prec(g)
3
sage: g.common_prec(f)
3

sage: f = t + t^2
sage: g = t^2
sage: f.common_prec(g)
+Infinity

sage: f = t^(-3) + O(t^(-2))
sage: g = t^(-5) + O(t^(-1))
sage: f.common_prec(g)
-2

sage: f = O(t^2)
sage: g = O(t^5)
sage: f.common_prec(g)
2
```

#### common_valuation(other)

Return the minimum valuation of `self` and `other`.

### EXAMPLES:

```
sage: R.<t> = LaurentSeriesRing(QQ)
sage: f = t^(-1) + t + t^2 + O(t^3)
sage: g = t + t^3 + t^4 + O(t^4)
sage: f.common_valuation(g)
-1
sage: g.common_valuation(f)
-1

sage: f = t + t^2 + O(t^3)
sage: g = t^(-3) + t^2
sage: f.common_valuation(g)
-3
sage: g.common_valuation(f)
-3
```
sage: f = t + t^2
sage: g = t^2
sage: f.common_valuation(g)
1

sage: f = t^(-3) + O(t^(-2))
sage: g = t^(-5) + O(t^(-1))
sage: f.common_valuation(g)
-5

sage: f = O(t^2)
sage: g = O(t^5)
sage: f.common_valuation(g)
+Infinity

**degree()**

Return the degree of a polynomial equivalent to this power series modulo big oh of the precision.

**EXAMPLES:**

```
sage: x = Frac(QQ[['x']]).0
sage: g = x^2 - x^4 + O(x^8)
sage: g.degree()
4
sage: g = -10/x^5 + x^2 - x^4 + O(x^8)
sage: g.degree()
4
sage: (x^-2 + O(x^0)).degree()
-2
```

**derivative(**args**)

The formal derivative of this Laurent series, with respect to variables supplied in args.

Multiple variables and iteration counts may be supplied; see documentation for the global derivative() function for more details.

**See also:**

_derivative()

**EXAMPLES:**

```
sage: R.<x> = LaurentSeriesRing(QQ)
sage: g = 1/x^10 - x + x^2 - x^4 + O(x^8)
sage: g.derivative()
-10*x^-11 - 1 + 2*x - 4*x^3 + O(x^7)
sage: g.derivative(x)
-10*x^-11 - 1 + 2*x - 4*x^3 + O(x^7)
sage: R.<t> = PolynomialRing(ZZ)
sage: S.<x> = LaurentSeriesRing(R)
sage: f = 2*t/x + (3*t^2 + 6*t)*x + O(x^2)
sage: f.derivative()
-2*t*x^-2 + (3*t^2 + 6*t) + O(x)
sage: f.derivative(x)
-2*t*x^-2 + (3*t^2 + 6*t) + O(x)
sage: f.derivative(t)
2*x^-1 + (6*t + 6)*x + O(x^2)
```
exponents()

Return the exponents appearing in self with nonzero coefficients.

EXAMPLES:

```sage
sage: R.<t> = LaurentSeriesRing(QQ)
sage: f = -5/t^2 + t + t^2 - 10/3*t^3
sage: f.exponents()
[-2, 1, 2, 3]
```

integral()

The formal integral of this Laurent series with 0 constant term.

EXAMPLES: The integral may or may not be defined if the base ring is not a field.

```sage
t = LaurentSeriesRing(ZZ, 't').0
sage: f = 2*t^-3 + 3*t^2 + O(t^4)
sage: f.integral()
-t^-2 + t^3 + O(t^5)
sage: f = t^3
sage: f.integral()
Traceback (most recent call last):
... ArithmeticError: Coefficients of integral cannot be coerced into the base ring
```

The integral of 1/t is \( \log(t) \), which is not given by a Laurent series:

```sage
t = Frac(QQ['t']).0
sage: f = -1/t^3 - 31/t + O(t^3)
sage: f.integral()
Traceback (most recent call last):
... ArithmeticError: The integral of is not a Laurent series, since \( t^{-1} \) has a nonzero coefficient.
```

Another example with just one negative coefficient:

```sage
A.<t> = QQ[]
sage: f = -2*t^(-4) + O(t^8)
sage: f.integral()
2/3*t^-3 + O(t^9)
sage: f.integral().derivative() == f
True
```

inverse()

Return the inverse of self, i.e., self^(-1).

EXAMPLES:

```sage
R.<t> = LaurentSeriesRing(ZZ)
sage: t.inverse()
t^-1
sage: (1-t).inverse()
1 + t + t^2 + t^3 + t^4 + t^5 + t^6 + t^7 + t^8 + ...
```

is_monomial()

Return True if this element is a monomial. That is, if self is \( x^n \) for some integer \( n \).
EXAMPLES:

```python
sage: k.<z> = LaurentSeriesRing(QQ, 'z')
sage: (30*z).is_monomial()
False
sage: k(1).is_monomial()
True
sage: (z+1).is_monomial()
False
sage: (z^-2909).is_monomial()
True
sage: (3*z^-2909).is_monomial()
False
```

```python
is_unit()

Return True if this is Laurent series is a unit in this ring.

EXAMPLES:

```python
sage: R.<t> = LaurentSeriesRing(QQ)
sage: (2+t).is_unit()
True
sage: f = 2+t^2+O(t^10); f.is_unit()
True
sage: 1/f
1/2 - 1/4*t^2 + 1/8*t^4 - 1/16*t^6 + 1/32*t^8 + O(t^10)
sage: R(0).is_unit()
False
sage: R.<s> = LaurentSeriesRing(ZZ)
sage: f = 2 + s^2 + O(s^10)
sage: f.is_unit()
False
sage: 1/f
Traceback (most recent call last):
... ValueError: constant term 2 is not a unit
```

ALGORITHM: A Laurent series is a unit if and only if its “unit part” is a unit.

```python
is_zero()

EXAMPLES:

```python
sage: x = Frac(QQ['x']).0
sage: f = 1/x + x + x^2 + 3*x^4 + O(x^7)
sage: f.is_zero()
0
sage: z = 0*f
sage: z.is_zero()
1
```

```python
laurent_polynomial()

Return the corresponding Laurent polynomial.

EXAMPLES:

```python
sage: R.<t> = LaurentSeriesRing(QQ)
sage: f = t^-3 + t + 7*t^2 + O(t^5)
sage: g = f.laurent_polynomial(); g
t^-3 + t + 7*t^2
```
sage: g.parent()
Univariate Laurent Polynomial Ring in t over Rational Field

**lift_to_precision** *(absprec=None)*

Return a congruent Laurent series with absolute precision at least `absprec`.

**INPUT:**

- `absprec` – an integer or None (default: None), the absolute precision of the result. If None, lifts to an exact element.

**EXAMPLES:**

```python
sage: A.<t> = LaurentSeriesRing(GF(5))
sage: x = t^(-1) + t^2 + O(t^5)
sage: x.lift_to_precision(10)
t^-1 + t^2 + O(t^10)
sage: x.lift_to_precision()
t^-1 + t^2
```

**list**()

**EXAMPLES:**

```python
sage: R.<t> = LaurentSeriesRing(QQ)
sage: f = -5/t^(2) + t + t^2 - 10/3*t^3
sage: f.list()
[-5, 0, 0, 1, 1, -10/3]
```

**nth_root** *(n, prec=None)*

Return the n-th root of this Laurent power series.

**INPUT:**

- `n` – integer
- `prec` – integer (optional) - precision of the result. Though, if this series has finite precision, then the result can not have larger precision.

**EXAMPLES:**

```python
sage: R.<x> = LaurentSeriesRing(QQ)
sage: (x^-2 + 1 + x).nth_root(2)
x^-1 + 0.5*x + 0.25*x^2 - ... - 19437/65536*x^18 + O(x^19)
sage: (x^-2 + 1 + x).nth_root(2)**2
x^-2 + 1 + x + O(x^18)
sage: j = j_invariant_qexp()
sage: q = j.parent().gen()
sage: j(q^3).nth_root(3)
q^-1 + 248*q^2 + 4124*q^5 + ... + O(q^29)
sage: (j(q^2) - 1728).nth_root(2)
q^-1 - 492*q - 22590*q^3 - ... + O(q^19)
```

**power_series**()

**EXAMPLES:**

```python
sage: R.<t> = LaurentSeriesRing(ZZ)
sage: f = 1/(1-t+O(t^10)); f.parent()
```

(continues on next page)
Laurent Series Ring in \( t \) over Integer Ring

```python
g = f.power_series(); g
1 + t + t^2 + t^3 + t^4 + t^5 + t^6 + t^7 + t^8 + t^9 + O(t^{10})
g = parent(g)
Power Series Ring in t over Integer Ring
```

```python
f = 3/t^2 + t^2 + t^3 + O(t^{10})
f.power_series()
Traceback (most recent call last):
...
TypeError: self is not a power series
```

**prec()**
This function returns the \( n \) so that the Laurent series is of the form \((\text{stuff}) + O(t^n)\). It doesn’t matter how many negative powers appear in the expansion. In particular, \( \text{prec} \) could be negative.

**EXAMPLES:**

```python
x = Frac(QQ[['x']]).0
f = x^2 + 3*x^4 + O(x^7)
f.prec()
7
g = 1/x^10 - x + x^2 - x^4 + O(x^8)
g.prec()
8
```

**precision_absolute()**
Return the absolute precision of this series.

By definition, the absolute precision of \( \ldots + O(x^r) \) is \( r \).

**EXAMPLES:**

```python
R.<t> = ZZ[[[]]
(t^2 + O(t^3)).precision_absolute()
3
(1 - t^2 + O(t^{100})).precision_absolute()
100
```

**precision_relative()**
Return the relative precision of this series, that is the difference between its absolute precision and its valuation.

By convention, the relative precision of \( 0 \) (or \( O(x^r) \) for any \( r \)) is 0.

**EXAMPLES:**

```python
R.<t> = ZZ[[[]]
(t^2 + O(t^3)).precision_relative()
1
(1 - t^2 + O(t^{100})).precision_relative()
100
O(t^4).precision_relative()
0
```

**residue()**
Return the residue of \( \text{self} \).
Consider the Laurent series
\[ f = \sum_{n \in \mathbb{Z}} a_n t^n = \cdots + \frac{a_{-2}}{t^2} + \frac{a_{-1}}{t} + a_0 + a_1 t + a_2 t^2 + \cdots, \]
then the residue of \( f \) is \( a_{-1} \). Alternatively this is the coefficient of \( 1/t \).

**EXAMPLES:**

```python
sage: t = LaurentSeriesRing(ZZ,'t').gen()
sage: f = 1/t**2+2/t+3+4*t
sage: f.residue()
2
sage: f = t+t**2
sage: f.residue()
0
sage: f.residue().parent()
Integer Ring
```

**shift** \((k)\)

Returns this Laurent series multiplied by the power \( t^n \). Does not change this series.

**Note:** Despite the fact that higher order terms are printed to the right in a power series, right shifting decreases the powers of \( t \), while left shifting increases them. This is to be consistent with polynomials, integers, etc.

**EXAMPLES:**

```python
sage: R.<t> = LaurentSeriesRing(QQ['y'])
sage: f = (t+t^-1)^4; f
\( t^{-4} + 4*t^{-2} + 6 + 4*t^2 + t^4 \)
sage: f.shift(10)
\( t^6 + 4*t^8 + 6*t^{10} + 4*t^{12} + t^{14} \)
sage: f >> 10
\( t^{-14} + 4*t^{-12} + 6*t^{-10} + 4*t^-8 + t^{-6} \)
sage: t << 4
\( t^5 \)
sage: t + O(t^3) >> 4
\( t^{-3} + O(t^{-1}) \)
```

**AUTHORS:**
- Robert Bradshaw (2007-04-18)

**truncate** \((n)\)

Return the Laurent series of degree ‘\(< n\)’ which is equivalent to self modulo \( x^n \).

**EXAMPLES:**

```python
sage: A.<x> = LaurentSeriesRing(ZZ)
sage: f = 1/(1-x)
sage: f
\( 1 + x + x^2 + x^3 + x^4 + x^5 + x^6 + x^7 + x^8 + x^9 + x^{10} + x^{11} + x^{12} + x^{13} + x^{14} + x^{15} + x^{16} + x^{17} + x^{18} + x^{19} + O(x^{20}) \)
sage: f.truncate(10)
\( 1 + x + x^2 + x^3 + x^4 + x^5 + x^6 + x^7 + x^8 + x^9 \)
```

**truncate_laurentseries** \((n)\)

Replace any terms of degree ‘\( \geq n \)’ by big oh.
EXAMPLES:

```python
sage: A.<x> = LaurentSeriesRing(ZZ)
sage: f = 1/(1-x)
sage: f
1 + x + x^2 + x^3 + x^4 + x^5 + x^6 + x^7 + x^8 + x^9 + x^10 + x^11 + x^12 + x^13 + x^14 + x^15 + x^16 + x^17 + x^18 + x^19 + O(x^20)
sage: f.truncate_laurentseries(10)
1 + x + x^2 + x^3 + x^4 + x^5 + x^6 + x^7 + x^8 + x^9 + O(x^10)
```

### truncate_neg (n)

Return the Laurent series equivalent to `self` except without any degree `n` terms.

This is equivalent to:

```python
self - self.truncate(n)
```

EXAMPLES:

```python
sage: A.<t> = LaurentSeriesRing(ZZ)
sage: f = 1/(1-t)
sage: f.truncate_neg(15)
t^15 + t^16 + t^17 + t^18 + t^19 + O(t^20)
```

### valuation()

EXAMPLES:

```python
sage: R.<x> = LaurentSeriesRing(QQ)
sage: f = 1/x + x^2 + 3*x^4 + O(x^7)
sage: g = 1 - x + x^2 - x^4 + O(x^8)
sage: f.valuation()
-1
sage: g.valuation()
0
```

Note that the valuation of an element undistinguishable from zero is infinite:

```python
sage: h = f - f; h
O(x^7)
sage: h.valuation()
+Infinity
```

### valuation_zero_part()

EXAMPLES:

```python
sage: x = Frac(QQ['x']).0
sage: f = x + x^2 + 3*x^4 + O(x^7)
sage: f/x
1 + x + 3*x^3 + O(x^6)
sage: f.valuation_zero_part()
1 + x + 3*x^3 + O(x^6)
sage: g = 1/x^7 - x + x^2 - x^4 + O(x^8)
sage: g.valuation_zero_part()
1 - x^8 + x^9 - x^11 + O(x^15)
```

### variable()

EXAMPLES:
```python
sage: x = Frac(QQ[['x']]).0
sage: f = 1/x + x^2 + 3*x^4 + O(x^7)
sage: f.variable()
'x'
```

```
sage.rings.laurent_series_ring_element.is_LaurentSeries(x)
```
INDICES AND TABLES

- Index
- Module Index
- Search Page
sage.rings.laurent_series_ring, 67
sage.rings.laurent_series_ring_element, 73
sage.rings.multi_power_series_ring, 43
sage.rings.multi_power_series_ring_element, 51
sage.rings.power_series_pari, 37
sage.rings.power_series_poly, 31
sage.rings.power_series_ring, 1
sage.rings.power_series_ring_element, 11
INDEX

A

add_bigoh() (sage.rings.laurent_series_ring_element.LaurentSeries method), 74
add_bigoh() (sage.rings.multi_power_series_ring_element.MPowerSeries method), 56
add_bigoh() (sage.rings.power_series_ring_element.PowerSeries method), 12

B

base_extend() (sage.rings.laurent_series_ring.LaurentSeriesRing method), 68
base_extend() (sage.rings.power_series_ring.PowerSeriesRing_generic method), 5
base_extend() (sage.rings.power_series_ring_element.PowerSeries method), 13
bigoh() (sage.rings.multi_power_series_ring.MPowerSeriesRing_generic method), 46

C

degree() (sage.rings.laurent_series_ring_element.LaurentSeries method), 76

D

default_prec() (sage.rings.laurent_series_ring.LaurentSeriesRing method), 69
degree() (sage.rings.laurent_series_ring_element.LaurentSeries method), 76
degree() (sage.rings.multi_power_series_ring_element.MPowerSeries method), 57
degree() (sage.rings.power_series_poly.PowerSeries_poly method), 31
degree() (sage.rings.power_series_ring_element.PowerSeries method), 15
derivative() (sage.rings.laurent_series_ring_element.LaurentSeries method), 76
derivative() (sage.rings.multi_power_series_ring_element.MPowerSeries method), 57
derivative() (sage.rings.power_series_ring_element.PowerSeries method), 15
dict() (sage.rings.multi_power_series_ring_element.MPowerSeries method), 57
dict() (sage.rings.power_series_pari.PowerSeries_pari method), 38
dict() (sage.rings.power_series_poly.PowerSeries_poly method), 31

E
egf() (sage.rings.multi_power_series_ring_element.MPowerSeries method), 57
egf_to_ogf() (sage.rings.power_series_ring_element.PowerSeries method), 16
Element (sage.rings.laurent_series_ring.LaurentSeriesRing attribute), 68
Element (sage.rings.multi_power_series_ring.MPowerSeriesRing_generic attribute), 46
exp() (sage.rings.multi_power_series_ring_element.MPowerSeries method), 58
exp() (sage.rings.power_series_ring_element.PowerSeries method), 16
exponents() (sage.rings.laurent_series_ring_element.LaurentSeries method), 76
exponents() (sage.rings.multi_power_series_ring_element.MPowerSeries method), 58
exponents() (sage.rings.power_series_ring_element.PowerSeries method), 17

F
fraction_field() (sage.rings.laurent_series_ring.LaurentSeriesRing method), 69
fraction_field() (sage.rings.power_series_ring.PowerSeriesRing_over_field method), 9

G
gen() (sage.rings.laurent_series_ring.LaurentSeriesRing method), 69
gen() (sage.rings.multi_power_series_ring.MPowerSeriesRing_generic method), 48
gen() (sage.rings.power_series_ring.PowerSeriesRing_generic method), 6

I
integral() (sage.rings.laurent_series_ring_element.LaurentSeries method), 77
integral() (sage.rings.multi_power_series_ring_element.MPowerSeries method), 59
integral() (sage.rings.power_series_pari.PowerSeries_pari method), 38
integral() (sage.rings.power_series_ring_element.MPowerSeries method), 32
inverse() (sage.rings.laurent_series_ring_element.LaurentSeries method), 77
inverse() (sage.rings.power_series_ring_element.PowerSeries method), 17
is_dense() (sage.rings.laurent_series_ring.LaurentSeriesRing method), 70
is_dense() (sage.rings.multi_power_series_ring_element.MPowerSeries method), 48
is_dense() (sage.rings.power_series_ring.Element method), 6
is_dense() (sage.rings.power_series_ring_element.PowerSeries method), 6
is_exact() (sage.rings.laurent_series_ring.LaurentSeriesRing method), 70
is_exact() (sage.rings.power_series_ring_element.PowerSeries method), 7
is_field() (sage.rings.laurent_series_ring.LaurentSeriesRing method), 70
is_field() (sage.rings.power_series_ring.Element method), 6
is_finite() (sage.rings.power_series_ring_element.PowerSeries method), 7
is_gen() (sage.rings.power_series_ring_element.PowerSeries method), 18
is_integral_domain() (sage.rings.multi_power_series_ring.MPowerSeriesRing_generic method), 48
is_LaurentSeries() (module sage.rings.laurent_series_ring), 83
is_LaurentSeriesRing() (module sage.rings.laurent_series_ring), 71
is_monomial() (sage.rings.laurent_series_ring_element.LaurentSeries method), 77
is_monomial() (sage.rings.power_series_ring_element.PowerSeries method), 18
is_MPowerSeries() (in module sage.rings.multi_power_series_ring_element), 66
is_MPowerSeriesRing() (in module sage.rings.multi_power_series_ring), 50
is_nilpotent() (sage.rings.multi_power_series_ring_element.MPowerSeries method), 60
is_noetherian() (sage.rings.multi_power_series_ring.MPowerSeriesRing_generic method), 48
is_PowerSeries() (in module sage.rings.power_series_ring_element), 28
is_PowerSeriesRing() (in module sage.rings.power_series_ring), 9
is_sparse() (sage.rings.laurent_series_ring.LaurentSeriesRing method), 70
is_sparse() (sage.rings.multi_power_series_ring.MPowerSeriesRing_generic method), 48
is_sparse() (sage.rings.power_series_ring.PowerSeriesRing_generic method), 7
is_sparse() (sage.rings.power_series_ring_element.PowerSeries method), 18
is_square() (sage.rings.multi_power_series_ring_element.MPowerSeries method), 60
is_square() (sage.rings.power_series_ring_element.PowerSeries method), 19
is_unit() (sage.rings.laurent_series_ring_element.LaurentSeries method), 78
is_unit() (sage.rings.multi_power_series_ring_element.MPowerSeries method), 60
is_unit() (sage.rings.power_series_ring_element.PowerSeries method), 19
is_zero() (sage.rings.laurent_series_ring_element.LaurentSeries method), 78

L

laurent_polynomial() (sage.rings.laurent_series_ring_element.LaurentSeries method), 78
laurent_polynomial_ring() (sage.rings.laurent_series_ring.LaurentSeriesRing method), 70
laurent_series() (sage.rings.multi_power_series_ring_element.MPowerSeries method), 61
laurent_series() (sage.rings.power_series_ring_element.PowerSeries method), 19
laurent_series_ring() (sage.rings.multi_power_series_ring.MPowerSeriesRing_generic method), 49
LaurentSeries (class in sage.rings.laurent_series_ring_element), 73
LaurentSeriesRing (class in sage.rings.laurent_series_ring), 67
lift_to_precision() (sage.rings.laurent_series_ring_element.LaurentSeries method), 79
lift_to_precision() (sage.rings.power_series_ring_element.PowerSeries method), 20
list() (sage.rings.laurent_series_ring_element.LaurentSeries method), 79
list() (sage.rings.multi_power_series_ring_element.MPowerSeries method), 61
list() (sage.rings.power_series_pari.PowerSeries_pari method), 38
list() (sage.rings.power_series_poly.PowerSeries_poly method), 32
list() (sage.rings.power_series_ring_element.PowerSeries method), 20
log() (sage.rings.multi_power_series_ring_element.MPowerSeries method), 61
log() (sage.rings.power_series_ring_element.PowerSeries method), 20

M

make_element_from_parent_v0() (in module sage.rings.power_series_ring_element), 29
make_powerseries_poly_v0() (in module sage.rings.power_series_poly), 36
make_powerseries_poly_v0() (in module sage.rings.power_series_ring_element), 29
MO (class in sage.rings.multi_power_series_ring_element), 53
monomials() (sage.rings.multi_power_series_ring_element.MPowerSeries method), 61
MPowerSeries (class in sage.rings.multi_power_series_ring_element), 54
MPowerSeriesRing_generic (class in sage.rings.multi_power_series_ring), 46

N

ngens() (sage.rings.laurent_series_ring.LaurentSeriesRing method), 70
ngens() (sage.rings.multi_power_series_ring.MPowerSeriesRing_generic method), 49
ngens() (sage.rings.power_series_ring.PowerSeriesRing_generic method), 7
nth_root() (sage.rings.laurent_series_ring_element.LaurentSeries method), 79
nth_root() (sage.rings.power_series_ring_element.PowerSeries method), 21

O

O() (sage.rings.laurent_series_ring_element.LaurentSeries method), 74
O() (sage.rings.multi_power_series_ring.MPowerSeriesRing_generic method), 46
O() (sage.rings.multi_power_series_ring_element.MPowerSeries method), 55
O() (sage.rings.power_series_ring_element.PowerSeries method), 12
ogf() (sage.rings.multi_power_series_ring_element.MPowerSeries method), 62
ogf_to_egf() (sage.rings.power_series_ring_element.PowerSeries method), 21

P

padded_list() (sage.rings.multi_power_series_ring_element.MPowerSeries method), 62
padded_list() (sage.rings.power_series_pari.PowerSeries_pari method), 39
padded_list() (sage.rings.power_series_ring_element.PowerSeries method), 22
pade() (sage.rings.power_series_poly.PowerSeries_poly method), 32
polynomial() (sage.rings.multi_power_series_ring_element.MPowerSeries method), 62
polynomial() (sage.rings.power_series_pari.PowerSeries_pari method), 39
polynomial() (sage.rings.power_series_poly.PowerSeries_poly method), 33
polynomial() (sage.rings.power_series_ring_element.PowerSeries method), 22
polynomial_ring() (sage.rings.laurent_series_ring.LaurentSeriesRing method), 70
power_series() (sage.rings.laurent_series_ring_element.LaurentSeries method), 79
power_series_ring() (sage.rings.laurent_series_ring.LaurentSeriesRing method), 71
PowerSeries (class in sage.rings.power_series_ring_element), 12
PowerSeries_pari (class in sage.rings.power_series_pari), 38
PowerSeries_poly (class in sage.rings.power_series_poly), 31
PowerSeriesRing() (in module sage.rings.power_series_ring), 2
PowerSeriesRing_domain (class in sage.rings.power_series_ring), 4
PowerSeriesRing_generic (class in sage.rings.power_series_ring), 4
PowerSeriesRing_over_field (class in sage.rings.power_series_ring), 9
prec() (sage.rings.laurent_series_ring_element.LaurentSeries method), 80
prec() (sage.rings.multi_power_series_ring_element.MPowerSeries method), 62
prec() (sage.rings.power_series_ring_element.PowerSeries method), 22
prec_ideal() (sage.rings.multi_power_series_ring_element.MPowerSeriesRing_generic method), 49
precision_absolute() (sage.rings.laurent_series_ring_element.LaurentSeries method), 80
precision_absolute() (sage.rings.power_series_ring_element.PowerSeries method), 23
precision_relative() (sage.rings.laurent_series_ring_element.LaurentSeries method), 80
precision_relative() (sage.rings.power_series_ring_element.PowerSeries method), 23

Q

quo_rem() (sage.rings.multi_power_series_ring_element.MPowerSeries method), 63

R

random_element() (sage.rings.power_series_ring.PowerSeriesRing_generic method), 7
remove_vart() (sage.rings.multi_power_series_ring_element.MPowerSeriesRing_generic method), 49
residue() (sage.rings.laurent_series_ring_element.LaurentSeries method), 80
residue_field() (sage.rings.laurent_series_ring.LaurentSeriesRing method), 71
residue_field() (sage.rings.power_series_ring.PowerSeriesRing_generic method), 8
reverse() (sage.rings.power_series_pari.PowerSeries_pari method), 39
reverse() (sage.rings.power_series_poly.PowerSeries_poly method), 33

S

sage.rings.laurent_series_ring (module), 67
sage.rings.laurent_series_ring_element (module), 73
sage.rings.multi_power_series_ring (module), 43
sage.rings.multi_power_series_ring_element (module), 51
sage.rings.power_series_pari (module), 37
sage.rings.power_series_poly (module), 31
sage.rings.power_series_ring (module), 1
sage.rings.power_series_ring_element (module), 11
shift() (sage.rings.laurent_series_ring_element.LaurentSeries method), 81
shift() (sage.rings.multi_power_series_ring_element.MPowerSeries method), 64
shift() (sage.rings.power_series_ring_element.PowerSeries method), 23
sin() (sage.rings.power_series_ring_element.PowerSeries method), 24
solve_linear_de() (sage.rings.multi_power_series_ring_element.MPowerSeries method), 64
sqrt() (sage.rings.multi_power_series_ring_element.MPowerSeries method), 64
sqrt() (sage.rings.power_series_ring_element.PowerSeries method), 25
square_root() (sage.rings.multi_power_series_ring_element.MPowerSeries method), 64
square_root() (sage.rings.power_series_ring_element.PowerSeries method), 27

T

term_order() (sage.rings.multi_power_series_ring.MPowerSeriesRing_generic method), 49
trailing_monomial() (sage.rings.multi_power_series_ring_element.MPowerSeries method), 64
truncate() (sage.rings.laurent_series_ring_element.LaurentSeries method), 81
truncate() (sage.rings.multi_power_series_ring_element.MPowerSeries method), 65
truncate() (sage.rings.power_series_poly.PowerSeries_poly method), 35
truncate() (sage.rings.power_series_ring_element.PowerSeries method), 27
truncate_laurentseries() (sage.rings.laurent_series_ring_element.LaurentSeries method), 81
truncate_neg() (sage.rings.laurent_series_ring_element.LaurentSeries method), 82
truncate_powerseries() (sage.rings.power_series_poly.PowerSeries_poly method), 35

U

uniformizer() (sage.rings.laurent_series_ring.LaurentSeriesRing method), 71
uniformizer() (sage.rings.power_series_ring.PowerSeriesRing_generic method), 8
unpickle_multi_power_series_ring_v0() (in module sage.rings.multi_power_series_ring), 50
unpickle_power_series_ring_v0() (in module sage.rings.power_series_ring), 9

V

V() (sage.rings.multi_power_series_ring_element.MPowerSeries method), 55
V() (sage.rings.power_series_ring_element.PowerSeries method), 12
valuation() (sage.rings.laurent_series_ring_element.LaurentSeries method), 82
valuation() (sage.rings.multi_power_series_ring_element.MPowerSeries method), 65
valuation() (sage.rings.power_series_pari.PowerSeries_pari method), 41
valuation() (sage.rings.power_series_poly.PowerSeries_poly method), 35
valuation() (sage.rings.power_series_ring_element.PowerSeries method), 27
valuation_zero_part() (sage.rings.laurent_series_ring_element.LaurentSeries method), 82
valuation_zero_part() (sage.rings.multi_power_series_ring_element.MPowerSeries method), 65
valuation_zero_part() (sage.rings.power_series_ring_element.PowerSeries method), 28
variable() (sage.rings.laurent_series_ring_element.LaurentSeries method), 82
variable() (sage.rings.multi_power_series_ring_element.MPowerSeries method), 66
variable() (sage.rings.power_series_ring_element.PowerSeries method), 28
variable_names_recursive() (sage.rings.power_series_ring.PowerSeriesRing_generic method), 9
variables() (sage.rings.multi_power_series_ring_element.MPowerSeries method), 66