CHAPTER ONE

MODULAR FORMS FOR ARITHMETIC GROUPS

1.1 Creating spaces of modular forms

EXAMPLES:

```python
sage: m = ModularForms(Gamma1(4),11)
sage: m
Modular Forms space of dimension 6 for Congruence Subgroup Gamma1(4) of weight 11 over Rational Field
sage: m.basis()
[q - 134*q^5 + O(q^6),
 q^2 + 80*q^5 + O(q^6),
 q^3 + 16*q^5 + O(q^6),
 q^4 - 4*q^5 + O(q^6),
 1 + 4092/50521*q^2 + 472384/50521*q^3 + 4194300/50521*q^4 + O(q^6),
 q + 1024*q^2 + 59048*q^3 + 1048576*q^4 + 9765626*q^5 + O(q^6)]
```

`sage.modular.modform.constructor.CuspForms(group=1, weight=2, base_ring=None, use_cache=True, prec=6)`

Create a space of cuspidal modular forms.

See the documentation for the ModularForms command for a description of the input parameters.

EXAMPLES:

```python
sage: CuspForms(11,2)
Cuspidal subspace of dimension 1 of Modular Forms space of dimension 2 for Congruence Subgroup Gamma0(11) of weight 2 over Rational Field
```

`sage.modular.modform.constructor.EisensteinForms(group=1, weight=2, base_ring=None, use_cache=True, prec=6)`

Create a space of Eisenstein modular forms.

See the documentation for the ModularForms command for a description of the input parameters.

EXAMPLES:

```python
sage: EisensteinForms(11,2)
Eisenstein subspace of dimension 1 of Modular Forms space of dimension 2 for Congruence Subgroup Gamma0(11) of weight 2 over Rational Field
```
sage.modular.modform.constructor.ModularForms(group=1, weight=2, base_ring=None, eis_only=False, use_cache=True, prec=6)

Create an ambient space of modular forms.

INPUT:

- **group** - A congruence subgroup or a Dirichlet character \(\varepsilon\).
- **weight** - int, the weight, which must be an integer \(\geq 1\).
- **base_ring** - the base ring (ignored if group is a Dirichlet character)
- **eis_only** - if True, compute only the Eisenstein part of the space. Only permitted (and only useful) in weight 1, where computing dimensions of cusp form spaces is expensive.

Create using the command \texttt{ModularForms(group, weight, base_ring)} where group could be either a congruence subgroup or a Dirichlet character.

EXAMPLES: First we create some spaces with trivial character:

\begin{verbatim}
sage: ModularForms(Gamma0(11),2).dimension()
sage: ModularForms(Gamma0(1),12).dimension()
\end{verbatim}

If we give an integer \(N\) for the congruence subgroup, it defaults to \(\Gamma_0(N)\):

\begin{verbatim}
sage: ModularForms(1,12).dimension()
sage: ModularForms(11,4)
\end{verbatim}

We create some spaces for \(\Gamma_1(N)\).

\begin{verbatim}
sage: ModularForms(Gamma1(13),2)
sage: ModularForms(Gamma1(7),k).dimension() for k in [2,3,4,5]
sage: ModularForms(Gamma1(5),11).dimension()
\end{verbatim}

We create a space with character:

\begin{verbatim}
sage: e = (DirichletGroup(13).0)^2
sage: e.order()
sage: M = ModularForms(e, 2);
M
sage: f = M.T(2).charpoly('x'); f
sage: f.factor()
\end{verbatim}

2 Chapter 1. Modular Forms for Arithmetic Groups
We can also create spaces corresponding to the groups $\Gamma_H(N)$ intermediate between $\Gamma_0(N)$ and $\Gamma_1(N)$:

```
sage: G = GammaH(30, [11])
sage: M = ModularForms(G, 2); M
sage: M.T(7).charpoly().factor()  # long time (7s on sage.math, 2011)
(x + 4) * x^2 * (x - 6)^4 * (x + 6)^4 * (x - 8)^7 * (x^2 + 4)
```

More examples of spaces with character:

```
sage: e = DirichletGroup(5, RationalField()).gen(); e
Dirichlet character modulo 5 of conductor 5 mapping 2 |--> -1
sage: m = ModularForms(e, 2); m
Modular Forms space of dimension 2, character [-1] and weight 2 over Rational Field
sage: m == loads(dumps(m))
True
sage: m.T(2).charpoly('x')
x^2 - 1
sage: m = ModularForms(e, 6); m.dimension()
4
sage: m.T(2).charpoly('x')
x^4 - 917*x^2 - 42284
```

This came up in a subtle bug (github issue #5923):

```
sage: ModularForms(gp(1), gap(12))
Modular Forms space of dimension 2 for Modular Group SL(2,Z) of weight 12 over␣˓→Rational Field
```

This came up in another bug (related to github issue #8630):

```
sage: chi = DirichletGroup(109, CyclotomicField(3)).0
sage: ModularForms(chi, 2, base_ring = CyclotomicField(15))
Modular Forms space of dimension 10, character [\zeta3 + 1] and weight 2 over␣˓→Cyclotomic Field of order 15 and degree 8
```

We create some weight 1 spaces. Here modular symbol algorithms do not work. In some small examples we can prove using Riemann–Roch that there are no cusp forms anyway, so the entire space is Eisenstein:

```
sage: M = ModularForms(Gamma1(11), 1); M
Modular Forms space of dimension 5 for Congruence Subgroup Gamma1(11) of weight 1␣˓→over Rational Field
sage: M.basis()
[1 + 22*q^5 + O(q^6),
 q + 4*q^5 + O(q^6),
 q^2 - 4*q^5 + O(q^6),
 q^3 - 5*q^5 + O(q^6),
 q^4 - 3*q^5 + O(q^6)
]
sage: M.cuspidal_subspace().basis()
[  
```
When this does not work (which happens as soon as the level is more than about 30), we use the Hecke stability algorithm of George Schaeffer:

```
sage: M = ModularForms(Gamma1(57), 1); M # long time
Modular Forms space of dimension 38 for Congruence Subgroup Gamma1(57) of weight 1 over Rational Field
sage: M.cuspidal_submodule().basis() # long time
[ q - q^4 + O(q^6), q^3 - q^4 + O(q^6) ]
```

The Eisenstein subspace in weight 1 can be computed quickly, without triggering the expensive computation of the cuspidal part:

```
sage: E = EisensteinForms(Gamma1(59), 1); E # indirect doctest
Eisenstein subspace of dimension 29 of Modular Forms space for Congruence Subgroup Gamma1(59) of weight 1 over Rational Field
sage: (E.0 + E.2).q_expansion(40)
1 + q^2 + 196*q^29 - 197*q^30 - q^31 + q^33 + q^34 + q^37 + q^38 - q^39 + O(q^40)
```

```sage.modular.modform.constructor.ModularForms_clear_cache()
Clear the cache of modular forms.
EXAMPLES:
```
```
sage: M = ModularForms(37,2)
sage: sage.modular.modform.constructor._cache == {}
False
sage: sage.modular.modform.constructor.ModularForms_clear_cache()
sage: sage.modular.modform.constructor._cache
{}
```

```sage.modular.modform.constructor.Newform(identifier, group=None, weight=2, base_ring=Rational Field, names=None)
INPUT:
  • identifier - a canonical label, or the index of the specific newform desired
  • group - the congruence subgroup of the newform
  • weight - the weight of the newform (default 2)
  • base_ring - the base ring
  • names - if the newform has coefficients in a number field, a generator name must be specified
EXAMPLES:
```
sage: Newform('67a', names='a')
q + 2*q^2 - 2*q^3 + 2*q^4 + 2*q^5 + O(q^6)
sage: Newform('67b', names='a')
q + a1*q^2 + (-a1 - 3)*q^3 + (-3*a1 - 3)*q^4 - 3*q^5 + O(q^6)

sage.modular.modform.constructor.Newforms(group, weight=2, base_ring=None, names=None)

Returns a list of the newforms of the given weight and level (or weight, level and character). These are calculated as $\text{Gal}(\overline{F}/F)$-orbits, where $F$ is the given base field.

INPUT:

- **group** - the congruence subgroup of the newform, or a Nebentypus character
- **weight** - the weight of the newform (default 2)
- **base_ring** - the base ring (defaults to $\mathbb{Q}$ for spaces without character, or the base ring of the character otherwise)
- **names** - if the newform has coefficients in a number field, a generator name must be specified

EXAMPLES:

sage: Newforms(11, 2)
[q - 2*q^2 - q^3 + 2*q^4 + q^5 + O(q^6)]
sage: Newforms(65, names='a')
[q - q^2 - 2*q^3 - q^4 - q^5 + O(q^6),
 q + a1*q^2 + (a1 + 1)*q^3 + (-2*a1 - 1)*q^4 + q^5 + O(q^6),
 q + a2*q^2 + (-a2 + 1)*q^3 + q^4 - q^5 + O(q^6)]

A more complicated example involving both a nontrivial character, and a base field that is not minimal for that character:

sage: K.<i> = QuadraticField(-1)
sage: chi = DirichletGroup(5, K)[1]
sage: len(Newforms(chi, 7, names='a'))
1
sage: x = polygen(K); L.<c> = K.extension(x^2 - 402*i)
sage: N = Newforms(chi, 7, base_ring = L); len(N)
2
sage: sorted([N[0][2], N[1][2]]) == sorted([1/2*c - 5/2*i - 5/2, -1/2*c - 5/2*i - 5/2])
True

sage.modular.modform.constructor.canonical_parameters(group, level, weight, base_ring)

Given a group, level, weight, and base_ring as input by the user, return a canonicalized version of them, where level is a Sage integer, group really is a group, weight is a Sage integer, and base_ring a Sage ring. Note that we can’t just get the level from the group, because we have the convention that the character for $\Gamma_1(N)$ is None (which makes good sense).

INPUT:

- **group** - int, long, Sage integer, group, Dirichlet character, or
- **level** - int, long, Sage integer, or group
- **weight** - coercible to Sage integer
- **base_ring** - commutative Sage ring

1.1. Creating spaces of modular forms
OUTPUT:

- **level** - Sage integer
- **group** - congruence subgroup
- **weight** - Sage integer
- **ring** - commutative Sage ring

EXAMPLES:

```python
sage: from sage.modular.modform.constructor import canonical_parameters
sage: v = canonical_parameters(5, 5, int(7), ZZ); v
(5, Congruence Subgroup Gamma0(5), 7, Integer Ring)
sage: type(v[0]), type(v[1]), type(v[2]), type(v[3])
(<class 'sage.rings.integer.Integer'>,
 <class 'sage.modular.arithgroup.congroup_gamma0.Gamma0_class_with_category'>,
 <class 'sage.rings.integer.Integer'>,
 <class 'sage.rings.integer_ring.IntegerRing_class'>)
sage: canonical_parameters(5, 7, 7, ZZ)
Traceback (most recent call last):
 ... ValueError: group and level do not match.
```

```
sage.modular.modform.constructor.parse_label(s)
```

Given a string `s` corresponding to a newform label, return the corresponding group and index.

EXAMPLES:

```
sage: sage.modular.modform.constructor.parse_label('11a')
(Congruence Subgroup Gamma0(11), 0)
sage: sage.modular.modform.constructor.parse_label('11aG1')
(Congruence Subgroup Gamma1(11), 0)
sage: sage.modular.modform.constructor.parse_label('11wG1')
(Congruence Subgroup Gamma(11), 22)
```

GammaH labels should also return the group and index (github issue #20823):

```
sage: sage.modular.modform.constructor.parse_label('389cGH[16]')
(Congruence Subgroup Gamma_H(389) with H generated by [16], 2)
```

### 1.2 Generic spaces of modular forms

EXAMPLES (computation of base ring): Return the base ring of this space of modular forms.

EXAMPLES: For spaces of modular forms for $\Gamma_0(N)$ or $\Gamma_1(N)$, the default base ring is $\mathbb{Q}$:

```
sage: ModularForms(11,2).base_ring()
Rational Field
sage: ModularForms(1,12).base_ring()
Rational Field
sage: CuspForms(Gamma1(13),3).base_ring()
Rational Field
```

The base ring can be explicitly specified in the constructor function.
For modular forms with character the default base ring is the field generated by the image of the character.

```
sage: ModularForms(DirichletGroup(13).0,3).base_ring()
Cyclotomic Field of order 12 and degree 4
```

For example, if the character is quadratic then the field is $\mathbb{Q}$ (if the characteristic is 0).

```
sage: ModularForms(DirichletGroup(13).0^6,3).base_ring()
Rational Field
```

An example in characteristic 7:

```
sage: ModularForms(13,3,base_ring=GF(7)).base_ring()
Finite Field of size 7
```

AUTHORS:


```python
class sage.modular.modform.space.ModularFormsSpace(group, weight, character, base_ring, category=None)

    Bases: HeckeModule_generic
    A generic space of modular forms.

    Element
global alias of ModularFormElement

def basis()
    Return a basis for self.

    EXAMPLES:
    sage: MM = ModularForms(11,2)
sage: MM.basis()
[[q - 2*q^2 - q^3 + 2*q^4 + q^5 + O(q^6),
  1 + 12/5*q + 36/5*q^2 + 48/5*q^3 + 84/5*q^4 + 72/5*q^5 + O(q^6)]

def character()
    Return the Dirichlet character corresponding to this space of modular forms. Returns None if there is no specific character corresponding to this space, e.g., if this is a space of modular forms on $\Gamma_1(N)$ with $N > 1$.

    EXAMPLES: The trivial character:
    sage: ModularForms(Gamma0(11),2).character()
    Dirichlet character modulo 11 of conductor 1 mapping 2 |---> 1
```

Spaces of forms with nontrivial character:
```
sage: ModularForms(DirichletGroup(20).0,3).character()
Dirichlet character modulo 20 of conductor 4 mapping 11 |--> -1, 17 |--> 1

sage: N = ModularForms(DirichletGroup(11).0, 3)
sage: N.character()
Dirichlet character modulo 11 of conductor 11 mapping 2 |--> zeta10

sage: s = N.cuspidal_submodule()
sage: s.character()
Dirichlet character modulo 11 of conductor 11 mapping 2 |--> zeta10

sage: CuspForms(DirichletGroup(11).0,3).character()
Dirichlet character modulo 11 of conductor 11 mapping 2 |--> zeta10

A space of forms with no particular character (hence None is returned):

```
sage: print(ModularForms(Gamma1(11),2).character())
None
```

If the level is one then the character is trivial.

```
sage: ModularForms(Gamma1(1),12).character()
Dirichlet character modulo 1 of conductor 1
```

cuspidal_submodule()

Return the cuspidal submodule of self.

EXAMPLES:

```
sage: N = ModularForms(6,4) ; N
Modular Forms space of dimension 5 for Congruence Subgroup Gamma0(6) of weight 4 over Rational Field

sage: N.eisenstein_subspace().dimension()
4

sage: N.cuspidal_submodule()
Cuspidal subspace of dimension 1 of Modular Forms space of dimension 5 for Congruence Subgroup Gamma0(6) of weight 4 over Rational Field

sage: N.cuspidal_submodule().dimension()
1
```

We check that a bug noticed on github issue #10450 is fixed:

```
sage: M = ModularForms(6, 10)
sage: W = M.span_of_basis(M.basis()[0:2])
sage: W.cuspidal_submodule()
Modular Forms subspace of dimension 2 of Modular Forms space of dimension 11 for Congruence Subgroup Gamma0(6) of weight 10 over Rational Field
```

cuspidal_subspace()

Synonym for cuspidal_submodule.

EXAMPLES:
```sage
N = ModularForms(6,4) ; N
Modular Forms space of dimension 5 for Congruence Subgroup Gamma0(6) of weight \rightarrow 4 over Rational Field
sage: N.eisenstein_subspace().dimension()
4

sage: N.cuspidal_subspace()
Cuspidal subspace of dimension 1 of Modular Forms space of dimension 5 for \rightarrow Congruence Subgroup Gamma0(6) of weight 4 over Rational Field
sage: N.cuspidal submodule().dimension()
1
```

decomposition()

This function returns a list of submodules \( V(f_i, t) \) corresponding to newforms \( f_i \) of some level dividing the level of self, such that the direct sum of the submodules equals self, if possible. The space \( V(f_i, t) \) is the image under \( g(q) \) maps to \( g(q^t) \) of the intersection with \( R[[q]] \) of the space spanned by the conjugates of \( f_i \), where \( R \) is the base ring of self.

TODO: Implement this function.

EXAMPLES:

```sage
sage: M = ModularForms(11,2); M.decomposition()
Traceback (most recent call last):...
NotImplementedError
```

echelon_basis()

Return a basis for self in reduced echelon form. This means that if we view the \( q \)-expansions of the basis as defining rows of a matrix (with infinitely many columns), then this matrix is in reduced echelon form.

EXAMPLES:

```sage
sage: M = ModularForms(Gamma0(11),4)
sage: M.echelon_basis()
[1 + O(q^6),
 q - 9*q^4 - 10*q^5 + O(q^6),
 q^2 + 6*q^4 + 12*q^5 + O(q^6),
 q^3 + q^4 + q^5 + O(q^6)]
sage: M.cuspidal_subspace().echelon_basis()
[q + 3*q^3 - 6*q^4 - 7*q^5 + O(q^6),
 q^2 - 4*q^3 + 2*q^4 + 8*q^5 + O(q^6)]
sage: M = ModularForms(SL2Z, 12)
sage: M.echelon_basis()
[1 + 196560*q^2 + 16773120*q^3 + 398034000*q^4 + 4629381120*q^5 + O(q^6),
 q - 24*q^2 + 252*q^3 - 1472*q^4 + 4830*q^5 + O(q^6)]
```

1.2. Generic spaces of modular forms
echelon_form()

Return a space of modular forms isomorphic to self but with basis of \( q \)-expansions in reduced echelon form.

This is useful, e.g., the default basis for spaces of modular forms is rarely in echelon form, but echelon form is useful for quickly recognizing whether a \( q \)-expansion is in the space.

EXAMPLES: We first illustrate two ambient spaces and their echelon forms.

\begin{verbatim}
    sage: M = ModularForms(11)
    sage: M.basis()
    [q - 2*q^2 - q^3 + 2*q^4 + q^5 + O(q^6),
     1 + 12/5*q + 36/5*q^2 + 48/5*q^3 + 84/5*q^4 + 72/5*q^5 + O(q^6)]
    sage: M.echelon_form().basis()
    [1 + 12*q^2 + 12*q^3 + 12*q^4 + 12*q^5 + O(q^6),
     q - 2*q^2 - q^3 + 2*q^4 + q^5 + O(q^6)]
    sage: M = ModularForms(Gamma1(6),4)
    sage: M.basis()
    [q - 2*q^2 - 3*q^3 + 4*q^4 + 6*q^5 + O(q^6),
     1 + O(q^6),
     q - 8*q^4 + 126*q^5 + O(q^6),
     q^2 + 9*q^4 + O(q^6),
     q^3 + O(q^6)]
    sage: M.echelon_form().basis()
    [1 + O(q^6),
     q + 94*q^5 + O(q^6),
     q^2 + 36*q^5 + O(q^6),
     q^3 + O(q^6),
     q^4 - 4*q^5 + O(q^6)]
\end{verbatim}

We create a space with a funny basis then compute the corresponding echelon form.

\begin{verbatim}
    sage: M = ModularForms(11,4)
    sage: M.basis()
    [q + 3*q^3 - 6*q^4 - 7*q^5 + O(q^6),
     q^2 - 4*q^3 + 2*q^4 + 8*q^5 + O(q^6),
     q^3 + O(q^6)]
    sage: M.echelon_form().basis()
    [q + 4*q^2 + 4*q^3 - 5*q^4 + 3*q^5 + O(q^6),
     q^2 - 4*q^3 + 2*q^4 + 8*q^5 + O(q^6),
     q^3 + O(q^6)]
\end{verbatim}


\begin{verbatim}
1 + 0(q^6),
q + 9*q^2 + 28*q^3 + 73*q^4 + 126*q^5 + 0(q^6)
\]
sage: F = M.span_of_basis([M.0 + 1/3*M.1, M.2 + M.3]); F.basis()
[ q + 1/3*q^2 + 5/3*q^3 - 16/3*q^4 - 13/3*q^5 + O(q^6),
1 + q + 9*q^2 + 28*q^3 + 73*q^4 + 126*q^5 + O(q^6)
\]
sage: E = F.echelon_form(); E.basis()
[ 1 + 26/3*q^2 + 79/3*q^3 + 235/3*q^4 + 391/3*q^5 + O(q^6),
q + 1/3*q^2 + 5/3*q^3 - 16/3*q^4 - 13/3*q^5 + O(q^6)
\]
\end{verbatim}

### eisenstein_series()

Compute the Eisenstein series associated to this space.

**Note:** This function should be overridden by all derived classes.

**EXAMPLES:**

\begin{verbatim}
sage: M = sage.modular.modform.space.ModularFormsSpace(Gamma0(11), 2, DirichletGroup(1)[0], base_ring=QQ); M.eisenstein_series()
Traceback (most recent call last):
  ... NotImplementedError: computation of Eisenstein series in this space not yet implemented
\end{verbatim}

### eisenstein_submodule()

Return the Eisenstein submodule for this space of modular forms.

**EXAMPLES:**

\begin{verbatim}
sage: M = ModularForms(11,2)
sage: M.eisenstein_submodule()
Eisenstein subspace of dimension 1 of Modular Forms space of dimension 2 for Congruence Subgroup Gamma0(11) of weight 2 over Rational Field
\end{verbatim}

We check that a bug noticed on [github issue #10450](https://github.com) is fixed:

\begin{verbatim}
sage: M = ModularForms(6, 10)
sage: W = M.span_of_basis(M.basis()[0:2])
sage: W.eisenstein_submodule()
Modular Forms subspace of dimension 0 of Modular Forms space of dimension 11 for Congruence Subgroup Gamma0(6) of weight 10 over Rational Field
\end{verbatim}

### eisenstein_subspace()

Synonym for eisenstein_submodule.

**EXAMPLES:**
```python
sage: M = ModularForms(11,2)
sage: M.eisenstein_subspace()
Eisenstein subspace of dimension 1 of Modular Forms space of dimension 2 for Congruence Subgroup Gamma0(11) of weight 2 over Rational Field

embedded_submodule()
Return the underlying module of self.

EXAMPLES:
```}

```python
sage: N = ModularForms(6,4)
sage: N.dimension()
5

sage: N.embedded_submodule()
Vector space of dimension 5 over Rational Field
```

```python
find_in_space(f, forms=None, prec=None, indep=True)
INPUT:
• f - a modular form or power series
• forms - (default: None) a specific list of modular forms or q-expansions.
• prec - if forms are given, compute with them to the given precision
• indep - (default: True) whether the given list of forms are assumed to form a basis.

OUTPUT: A list of numbers that give f as a linear combination of the basis for this space or of the given forms if independent=True.

Note: If the list of forms is given, they do not have to be in self.

EXAMPLES:
```}

```python
sage: M = ModularForms(11,2)
sage: N = ModularForms(10,2)
sage: M.find_in_space( M.basis()[0] )
[1, 0]
sage: M.find_in_space( N.basis()[0], forms=N.basis() )
[1, 0, 0]
sage: M.find_in_space( N.basis()[0] )
Traceback (most recent call last):
  ... ArithmeticError: vector is not in free module
```

```python
gen(n)
Return the nth generator of self.

EXAMPLES:
```
sage: N = ModularForms(6,4)
sage: N.basis()
[ q - 2*q^2 - 3*q^3 + 4*q^4 + 6*q^5 + O(q^6),
  1 + O(q^6),
  q - 8*q^4 + 126*q^5 + O(q^6),
  q^2 + 9*q^4 + 0(q^6),
  q^3 + O(q^6) ]

sage: N.gen(0)
q - 2*q^2 - 3*q^3 + 4*q^4 + 6*q^5 + O(q^6)

sage: N.gen(4)
q^3 + O(q^6)

sage: N.gen(5)
Traceback (most recent call last):
  ... ValueError: Generator 5 not defined

gens()
Return a complete set of generators for self.

EXAMPLES:

sage: N = ModularForms(6,4)
sage: N.gens()
[ q - 2*q^2 - 3*q^3 + 4*q^4 + 6*q^5 + O(q^6),
  1 + O(q^6),
  q - 8*q^4 + 126*q^5 + O(q^6),
  q^2 + 9*q^4 + 0(q^6),
  q^3 + O(q^6) ]

group()
Return the congruence subgroup associated to this space of modular forms.

EXAMPLES:

sage: ModularForms(Gamma0(12),4).group()
Congruence Subgroup Gamma0(12)

sage: CuspForms(Gamma1(113),2).group()
Congruence Subgroup Gamma1(113)

Note that \( \Gamma_1(1) \) and \( \Gamma_0(1) \) are replaced by \( SL_2(\mathbb{Z}) \).

sage: CuspForms(Gamma1(1),12).group()
Modular Group SL(2,Z)
sage: CuspForms(SL2Z,12).group()
Modular Group SL(2,Z)
has_character()
Return True if this space of modular forms has a specific character.
This is True exactly when the character() function does not return None.

EXAMPLES: A space for $\Gamma_0(N)$ has trivial character, hence has a character.

```
sage: CuspForms(Gamma0(11),2).has_character()
True
```
A space for $\Gamma_1(N)$ (for $N \geq 2$) never has a specific character.

```
sage: CuspForms(Gamma1(11),2).has_character()
False
sage: CuspForms(DirichletGroup(11).0,3).has_character()
True
```

integral_basis()
Return an integral basis for this space of modular forms.

EXAMPLES:
In this example the integral and echelon bases are different.

```
sage: m = ModularForms(97,2,prec=10)
sage: s = m.cuspidal_subspace()
sage: s.integral_basis()
[ q + 2*q^7 + 4*q^8 - 2*q^9 + O(q^10),
q^2 + q^4 + q^7 + 3*q^8 - 3*q^9 + 0(q^10),
q^3 + q^4 - 3*q^8 + q^9 + 0(q^10),
2*q^4 - 2*q^8 + 0(q^10),
q^5 - 2*q^8 + 2*q^9 + O(q^10),
q^6 + 2*q^7 + 5*q^8 - 5*q^9 + O(q^10),
3*q^7 + 6*q^8 - 4*q^9 + O(q^10) ]
sage: s.echelon_basis()
[ q + 2/3*q^9 + O(q^10),
q^2 + 2*q^8 - 5/3*q^9 + O(q^10),
q^3 - 2*q^8 + q^9 + 0(q^10),
q^4 - q^8 + 0(q^10),
q^5 - 2*q^8 + 2*q^9 + O(q^10),
q^6 + q^8 - 7/3*q^9 + 0(q^10),
q^7 + 2*q^8 - 4/3*q^9 + O(q^10) ]
```
Here’s another example where there is a big gap in the valuations:

```
sage: m = CuspForms(64,2)
sage: m.integral_basis()
[ q + 0(q^6),
q^2 + 0(q^6),
q^5 + 0(q^6) ]
```
**is_ambient()**

Return True if this an ambient space of modular forms.

EXAMPLES:

```python
sage: M = ModularForms(Gamma1(4),4)
sage: M.is_ambient()
True
```

```python
sage: E = M.eisenstein_subspace()
sage: E.is_ambient()
False
```

**is_cuspidal()**

Return True if this space is cuspidal.

EXAMPLES:

```python
sage: M = ModularForms(Gamma0(11), 2).new_submodule()
sage: M.is_cuspidal()
False
```

```python
sage: M.cuspidal_submodule().is_cuspidal()
True
```

**is_eisenstein()**

Return True if this space is Eisenstein.

EXAMPLES:

```python
sage: M = ModularForms(Gamma0(11), 2).new_submodule()
```

```python
sage: M.is_eisenstein()  # False
```

```python
sage: M.eisenstein_submodule().is_eisenstein()  # True
```

**level()**

Return the level of self.

EXAMPLES:

```python
sage: M = ModularForms(47,3)
sage: M.level()
47
```

**modular_symbols**(sign=0)

Return the space of modular symbols corresponding to self with the given sign.

Note: This function should be overridden by all derived classes.

EXAMPLES:

```python
sage: M = sage.modular.modform.space.ModularFormsSpace(Gamma0(11), 2,\n˓→DirichletGroup(1)[0], base_ring=QQ); M.modular_symbols()
Traceback (most recent call last):
```

(continues on next page)
new_submodule\((p=\text{None})\)

Return the new submodule of self. If \(p\) is specified, return the \(p\)-new submodule of self.

**Note:** This function should be overridden by all derived classes.

**EXAMPLES:**

```python
sage: M = sage.modular.modform.space.ModularFormsSpace(Gamma0(11), 2, DirichletGroup(1)[0], base_ring=QQ); M.new_submodule()
Traceback (most recent call last):
...  
NotImplementedError: computation of new submodule not yet implemented
```

new_subspace\((p=\text{None})\)

Synonym for new_submodule.

**EXAMPLES:**

```python
sage: M = sage.modular.modform.space.ModularFormsSpace(Gamma0(11), 2, DirichletGroup(1)[0], base_ring=QQ); M.new_subspace()
Traceback (most recent call last):
...  
NotImplementedError: computation of new submodule not yet implemented
```

newforms\((\text{name}=\text{None})\)

Return all newforms in the cuspidal subspace of self.

**EXAMPLES:**

```python
sage: CuspForms(18,4).newforms()  
[q + 2q^2 + 4q^4 - 6q^5 + O(q^6)]

sage: CuspForms(32,4).newforms()
[q - 8q^3 - 10q^5 + O(q^6), q + 22q^5 + O(q^6), q + 8q^5 + 10q^5 + O(q^6)]

sage: CuspForms(23).newforms('b')
[q + b0q^2 + (-2b0 - 1)^q^3 + (-b0 - 1)^q^4 + 2b0q^q^5 + O(q^6)]

sage: CuspForms(23).newforms()
Traceback (most recent call last):
...  
ValueError: Please specify a name to be used when generating names for generators of Hecke eigenvalue fields corresponding to the newforms.
```

prec\((\text{new_prec}=\text{None})\)

Return or set the default precision used for displaying \(q\)-expansions of elements of this space.

**INPUT:**

- \text{new_prec} - positive integer (default: None)

**OUTPUT:** if \text{new_prec} is None, returns the current precision.

**EXAMPLES:**
```python
sage: M = ModularForms(1,12)
sage: S = M.cuspidal_subspace()
sage: S.prec()
6
sage: S.basis()
[ q - 24*q^2 + 252*q^3 - 1472*q^4 + 4830*q^5 + O(q^6) ]
sage: S.prec(8)
8
sage: S.basis()
[ q - 24*q^2 + 252*q^3 - 1472*q^4 + 4830*q^5 - 6048*q^6 - 16744*q^7 + O(q^8) ]
```

**q_echelon_basis**(prec=None)

Return the echelon form of the basis of \( q \)-expansions of self up to precision prec.

The \( q \)-expansions are power series (not actual modular forms). The number of \( q \)-expansions returned equals the dimension.

**EXAMPLES:**

```python
sage: M = ModularForms(11,2)
sage: M.q_expansion_basis()
[ q - 2*q^2 - q^3 + 2*q^4 + q^5 + O(q^6),
  1 + 12/5*q + 36/5*q^2 + 48/5*q^3 + 84/5*q^4 + 72/5*q^5 + O(q^6) ]
sage: M.q_echelon_basis()
[ 1 + 12*q^2 + 12*q^3 + 12*q^4 + 12*q^5 + O(q^6),
  q - 2*q^2 - q^3 + 2*q^4 + q^5 + O(q^6) ]
```

**q_expansion_basis**(prec=None)

Return a sequence of \( q \)-expansions for the basis of this space computed to the given input precision.

**INPUT:**

- prec - integer (\( \geq 0 \)) or None

If prec is None, the prec is computed to be at least large enough so that each \( q \)-expansion determines the form as an element of this space.

**Note:** In fact, the \( q \)-expansion basis is always computed to at least self.prec().

**EXAMPLES:**

```python
sage: S = ModularForms(11,2).cuspidal_submodule()
sage: S.q_expansion_basis()
[ q - 2*q^2 - q^3 + 2*q^4 + q^5 + O(q^6) ]
```
sage: S.q_expansion_basis(5)
\[q - 2*q^2 - q^3 + 2*q^4 + O(q^5)\]
sage: S = ModularForms(1,24).cuspidal_submodule()
sage: S.q_expansion_basis(8)
\[
q + 195660*q^3 + 12080128*q^4 + 44656110*q^5 - 982499328*q^6 - 147247240*q^7 + O(q^8),
q^2 - 48*q^3 + 1080*q^4 - 15040*q^5 + 143820*q^6 - 985824*q^7 + O(q^8)
\]

An example which used to be buggy:

sage: M = CuspForms(128, 2, prec=3)
sage: M.q_expansion_basis()
\[
q - q^{17} + O(q^{22}),
q^2 - 3*q^{18} + O(q^{22}),
q^3 - q^{11} + q^{19} + O(q^{22}),
q^4 - 2*q^{20} + O(q^{22}),
q^5 - 3*q^{21} + O(q^{22}),
q^7 - q^{15} + O(q^{22}),
q^9 - q^{17} + O(q^{22}),
q^{10} + O(q^{22}),
q^{13} - q^{21} + O(q^{22})
\]

\textbf{q_integral_basis}(\textit{prec=None})

Return a $\mathbb{Z}$-reduced echelon basis of $q$-expansions for self.

The $q$-expansions are power series with coefficients in $\mathbb{Z}$; they are \textit{not} actual modular forms.

The base ring of self must be $\mathbb{Q}$. The number of $q$-expansions returned equals the dimension.

\textbf{EXAMPLES:}

sage: S = CuspForms(11,2)
sage: S.q_integral_basis(5)
\[
q - 2*q^2 - q^3 + 2*q^4 + O(q^5)
\]

\textbf{set_precision}(\textit{new_prec})

Set the default precision used for displaying $q$-expansions.

\textbf{INPUT:}

\begin{itemize}
  \item \textit{new_prec} - positive integer
\end{itemize}

\textbf{EXAMPLES:}

sage: M = ModularForms(Gamma0(37),2)
sage: M.set_precision(10)
sage: S = M.cuspidal_subspace()
sage: S.basis()
[ q + q^3 - 2*q^4 - q^7 - 2*q^9 + O(q^10),
  q^2 + 2*q^3 - 2*q^4 + q^5 - 3*q^6 - 4*q^9 + O(q^10) ]

sage: S.set_precision(0)
sage: S.basis()
[ O(q^0),
  O(q^0) ]

The precision of subspaces is the same as the precision of the ambient space.

sage: S.set_precision(2)
sage: M.basis()
[ q + O(q^2),
  O(q^2),
  1 + 2/3*q + O(q^2) ]

The precision must be nonnegative:

sage: S.set_precision(-1)
Traceback (most recent call last):
 ... ValueError: n (=−1) must be >= 0

We do another example with nontrivial character.

sage: M = ModularForms(DirichletGroup(13).0^2)
sage: M.set_precision(10)
sage: M.cuspidal_subspace().0
q + (-zeta6 - 1)*q^2 + (2*zeta6 - 2)*q^3 + zeta6*q^4 + (-2*zeta6 + 1)*q^5 + (-
  2*zeta6 + 4)*q^6 + (2*zeta6 - 1)*q^8 - zeta6*q^9 + O(q^10)

\textbf{span}(B)

Take a set B of forms, and return the subspace of self with B as a basis.

\textbf{EXAMPLES:}

sage: N = ModularForms(6,4) ; N
Modular Forms space of dimension 5 for Congruence Subgroup Gamma0(6) of weight 4 over Rational Field

sage: N.span_of_basis([N.basis()][0])
Modular Forms subspace of dimension 1 of Modular Forms space of dimension 5 for Congruence Subgroup Gamma0(6) of weight 4 over Rational Field
sage: N.span_of_basis([N.basis()[0], N.basis()[1]])
Modular Forms subspace of dimension 2 of Modular Forms space of dimension 5 for...
→Congruence Subgroup Gamma0(6) of weight 4 over Rational Field

sage: N.span_of_basis( N.basis() )
Modular Forms subspace of dimension 5 of Modular Forms space of dimension 5 for...
→Congruence Subgroup Gamma0(6) of weight 4 over Rational Field

span_of_basis($B$)
Take a set $B$ of forms, and return the subspace of self with $B$ as a basis.

EXAMPLES:

sage: N = ModularForms(6,4) ; N
Modular Forms space of dimension 5 for Congruence Subgroup Gamma0(6) of weight...
→4 over Rational Field

sage: N.span_of_basis([N.basis()[0]])
Modular Forms subspace of dimension 1 of Modular Forms space of dimension 5 for...
→Congruence Subgroup Gamma0(6) of weight 4 over Rational Field

sage: N.span_of_basis([N.basis()[0], N.basis()[1]])
Modular Forms subspace of dimension 2 of Modular Forms space of dimension 5 for...
→Congruence Subgroup Gamma0(6) of weight 4 over Rational Field

sage: N.span_of_basis( N.basis() )
Modular Forms subspace of dimension 5 of Modular Forms space of dimension 5 for...
→Congruence Subgroup Gamma0(6) of weight 4 over Rational Field

sturm_bound($M=None$)
For a space $M$ of modular forms, this function returns an integer $B$ such that two modular forms in either
self or $M$ are equal if and only if their $q$-expansions are equal to precision $B$ (note that this is 1+ the usual
Sturm bound, since $O(q^{\text{prec}})$ has precision $\text{prec}$). If $M$ is none, then $M$ is set equal to self.

EXAMPLES:

sage: S37=CuspForms(37,2)

sage: S37.sturm_bound()
8

sage: M = ModularForms(11,2)

sage: M.sturm_bound()
3

sage: ModularForms(Gamma1(15),2).sturm_bound()
33

sage: CuspForms(Gamma1(144), 3).sturm_bound()
3457

sage: CuspForms(DirichletGroup(144).1^2, 3).sturm_bound()
73

sage: CuspForms(Gamma0(144), 3).sturm_bound()
73
NOTE:

Kevin Buzzard pointed out to me (William Stein) in Fall 2002 that the above bound is fine for Gamma with character, as one sees by taking a power of $f$. More precisely, if $f \equiv 0 \pmod{p}$ for first $s$ coefficients, then $f^r \equiv 0 \pmod{p}$ for first $sr$ coefficients. Since the weight of $f^r$ is $r \cdot \text{weight}(f)$, it follows that if $s \geq$ the Sturm bound for $\Gamma_0$ at weight($f$), then $f^r$ has valuation large enough to be forced to be $0$ at $r \cdot \text{weight}(f)$ by Sturm bound (which is valid if we choose $r$ right). Thus $f \equiv 0 \pmod{p}$. Conclusion: For $\Gamma_1$ with fixed character, the Sturm bound is exactly the same as for $\Gamma_0$. A key point is that we are finding $\mathbb{Z}[\epsilon]$ generators for the Hecke algebra here, not $\mathbb{Z}$-generators. So if one wants generators for the Hecke algebra over $\mathbb{Z}$, this bound is wrong.

This bound works over any base, even a finite field. There might be much better bounds over $\mathbb{Q}$, or for comparing two eigenforms.

weight()

Return the weight of this space of modular forms.

EXAMPLES:

\begin{verbatim}
sage: M = ModularForms(Gamma1(13),11)
sage: M.weight()
11

sage: M = ModularForms(Gamma0(997),100)
sage: M.weight()
100

sage: M = ModularForms(Gamma0(97),4)
sage: M.weight()
4
sage: M.eisenstein_submodule().weight()
4
\end{verbatim}

sage.modular.modform.space.contains_each($V$, $B$)

Determine whether or not $V$ contains every element of $B$. Used here for linear algebra, but works very generally.

EXAMPLES:

\begin{verbatim}
sage: contains_each = sage.modular.modform.space.contains_each
sage: contains_each( range(20), prime_range(20) )
True
sage: contains_each( range(20), range(30) )
False
\end{verbatim}

sage.modular.modform.space.is_ModularFormsSpace($x$)

Return True if $x$ is a `ModularFormsSpace`.

EXAMPLES:

\begin{verbatim}
sage: from sage.modular.modform.space import is_ModularFormsSpace
sage: is_ModularFormsSpace(ModularForms(11,2))
True
sage: is_ModularFormsSpace(CuspForms(11,2))
True
\end{verbatim}
1.3 Ambient spaces of modular forms

EXAMPLES:

We compute a basis for the ambient space \( M_2(\Gamma_1(25), \chi) \), where \( \chi \) is quadratic.

```python
sage: chi = DirichletGroup(25,QQ).0; chi
Dirichlet character modulo 25 of conductor 5 mapping 2 |--> -1
sage: n = ModularForms(chi,2); n
Modular Forms space of dimension 6, character [-1] and weight 2 over Rational Field
sage: type(n)
<class 'sage.modular.modform.ambient_eps.ModularFormsAmbient_eps_with_category'>
```

Compute a basis:

```python
sage: n.basis()
[1 + O(q^6), q + O(q^6), q^2 + O(q^6), q^3 + O(q^6), q^4 + O(q^6), q^5 + O(q^6)]
```

Compute the same basis but to higher precision:

```python
sage: n.set_precision(20)
sage: n.basis()
[1 + 10*q^10 + 20*q^15 + O(q^20), q + 5*q^6 + q^9 + 12*q^11 - 3*q^14 + 17*q^16 + 8*q^19 + O(q^20), q^2 + 4*q^7 - q^8 + 8*q^12 + 2*q^13 + 10*q^17 - 5*q^18 + O(q^20), q^3 + q^7 + 3*q^8 - q^12 + 5*q^13 + 3*q^17 + 6*q^18 + O(q^20), q^4 - q^6 + 2*q^9 + 3*q^14 - 2*q^16 + 4*q^19 + O(q^20), q^5 + q^10 + 2*q^15 + O(q^20)]
```

class sage.modular.modform.ambient.ModularFormsAmbient(group, weight, base_ring, character=None, eis_only=False)

Bases: ModularFormsSpace, AmbientHeckeModule

An ambient space of modular forms.

ambient_space()

Return the ambient space that contains this ambient space. This is, of course, just this space again.

EXAMPLES:
sage: m = ModularForms(Gamma0(3),30)
sage: m.ambient_space() is m
True

change_ring(base_ring)
Change the base ring of this space of modular forms.

INPUT:
• R - ring

EXAMPLES:

sage: M = ModularForms(Gamma0(37),2)
sage: M.basis()
[q + q^3 - 2*q^4 + O(q^6),
q^2 + 2*q^3 - 2*q^4 + q^5 + O(q^6),
1 + 2/3*q + 2*q^2 + 8/3*q^3 + 14/3*q^4 + 4*q^5 + O(q^6)]

The basis after changing the base ring is the reduction modulo 3 of an integral basis.

sage: M3 = M.change_ring(GF(3))
sage: M3.basis()
[q + q^3 + q^4 + O(q^6),
q^2 + 2*q^3 + q^4 + q^5 + O(q^6),
1 + q^3 + q^4 + 2*q^5 + O(q^6)]

cuspidal_submodule()
Return the cuspidal submodule of this ambient module.

EXAMPLES:

sage: ModularForms(Gamma1(13)).cuspidal_submodule()
Cuspidal subspace of dimension 2 of Modular Forms space of dimension 13 for Congruence Subgroup Gamma1(13) of weight 2 over Rational Field

dimension()
Return the dimension of this ambient space of modular forms, computed using a dimension formula (so it should be reasonably fast).

EXAMPLES:

sage: m = ModularForms(Gamma1(20),20)
sage: m.dimension()
238

eisenstein_params()
Return parameters that define all Eisenstein series in self.

OUTPUT: an immutable Sequence

EXAMPLES:
sage: m = ModularForms(Gamma0(22), 2)
sage: v = m.eisenstein_params(); v
[(Dirichlet character modulo 22 of conductor 1 mapping 13 \rightarrow 1, Dirichlet character modulo 22 of conductor 1 mapping 13 \rightarrow 1, 2), (Dirichlet character modulo 22 of conductor 1 mapping 13 \rightarrow 1, 11), (Dirichlet character modulo 22 of conductor 1 mapping 13 \rightarrow 1, 22)]
sage: type(v)
<class 'sage.structure.sequence.Sequence_generic'>

\textbf{eisenstein\_series()}

Return all Eisenstein series associated to this space.

```
sage: ModularForms(27,2).eisenstein_series()
[ q^3 + O(q^6),
 q - 3q^2 + 7q^4 - 6q^5 + O(q^6),
 1/12 + q + 3q^2 + 7q^4 + 6q^5 + O(q^6),
 1/3 + q + 3q^2 + 4q^3 + 7q^4 + 6q^5 + O(q^6),
 13/12 + q + 3q^2 + 4q^3 + 7q^4 + 6q^5 + O(q^6)
 ]
```

```
sage: ModularForms(Gamma1(5),3).eisenstein_series()
[ -1/5\zeta4 - 2/5 + q + (4\zeta4 + 1)q^2 + (-9\zeta4 + 1)q^3 + (4\zeta4 + 1)q^4 + q^5 + O(q^6),
 q + (zeta4 + 4)q^2 + (-zeta4 + 9)q^3 + (4\zeta4 + 15)q^4 + 25q^5 + O(q^6),
 1/5\zeta4 - 2/5 + q + (-4\zeta4 + 1)q^2 + (9\zeta4 + 1)q^3 + (-4\zeta4 + 15)q^4 + q^5 + O(q^6),
 q + (-zeta4 + 4)q^2 + (zeta4 + 9)q^3 + (-4\zeta4 + 15)q^4 + 25q^5 + O(q^6)
 ]
```

```
sage: eps = DirichletGroup(13).0^2
sage: ModularForms(eps,2).eisenstein_series()
[ -7/13\zeta6 - 11/13 + q + (2\zeta6 + 1)q^2 + (-3\zeta6 + 1)q^3 + (6\zeta6 - 3)q^4 - 4q^5 + O(q^6),
 q + (zeta6 + 2)q^2 + (-zeta6 + 3)q^3 + (3\zeta6 + 3)q^4 + 4q^5 + O(q^6)
 ]
```

\textbf{eisenstein\_submodule()}

Return the Eisenstein submodule of this ambient module.

\textbf{EXAMPLES:}

```
sage: m = ModularForms(Gamma1(13),2); m
Modular Forms space of dimension 13 for Congruence Subgroup Gamma1(13) of weight 2 over Rational Field
sage: m.eisenstein_submodule()
Eisenstein subspace of dimension 11 of Modular Forms space of dimension 13 for Congruence Subgroup Gamma1(13) of weight 2 over Rational Field
```
**free_module()**

Return the free module underlying this space of modular forms.

EXAMPLES:

```
sage: ModularForms(37).free_module()
Vector space of dimension 3 over Rational Field
```

**hecke_module_of_level(N)**

Return the Hecke module of level $N$ corresponding to self, which is the domain or codomain of a degeneracy map from self. Here $N$ must be either a divisor or a multiple of the level of self.

EXAMPLES:

```
sage: ModularForms(25, 6).hecke_module_of_level(5)
Modular Forms space of dimension 3 for Congruence Subgroup Gamma0(5) of weight 6 over Rational Field
sage: ModularForms(Gamma1(4), 3).hecke_module_of_level(8)
Modular Forms space of dimension 7 for Congruence Subgroup Gamma1(8) of weight 3 over Rational Field
sage: ModularForms(Gamma1(4), 3).hecke_module_of_level(9)
Traceback (most recent call last):
  ... ValueError: N (=9) must be a divisor or a multiple of the level of self (=4)
```

**hecke_polynomial**(n, var='x')

Compute the characteristic polynomial of the Hecke operator $T_n$ acting on this space. Except in level 1, this is computed via modular symbols, and in particular is faster to compute than the matrix itself.

EXAMPLES:

```
sage: ModularForms(17,4).hecke_polynomial(2)
x^6 - 16*x^5 + 18*x^4 + 608*x^3 - 1371*x^2 - 4968*x + 7776
```

Check that this gives the same answer as computing the actual Hecke matrix (which is generally slower):

```
sage: ModularForms(17,4).hecke_matrix(2).charpoly()
x^6 - 16*x^5 + 18*x^4 + 608*x^3 - 1371*x^2 - 4968*x + 7776
```

**is_ambient()**

Return True if this an ambient space of modular forms.

This is an ambient space, so this function always returns True.

EXAMPLES:

```
sage: ModularForms(11).is_ambient()
True
sage: CuspForms(11).is_ambient()
False
```

**modular_symbols**(sign=0)

Return the corresponding space of modular symbols with the given sign.

EXAMPLES:
```python
sage: S = ModularForms(11,2)
sage: S.modular_symbols()
Modular Symbols space of dimension 3 for Gamma_0(11) of weight 2 with sign 0 → over Rational Field
sage: S.modular_symbols(sign=1)
Modular Symbols space of dimension 2 for Gamma_0(11) of weight 2 with sign 1 → over Rational Field
sage: S.modular_symbols(sign=-1)
Modular Symbols space of dimension 1 for Gamma_0(11) of weight 2 with sign -1 → over Rational Field

sage: ModularForms(1,12).modular_symbols()
Modular Symbols space of dimension 3 for Gamma_0(1) of weight 12 with sign 0 → over Rational Field
```

**module()**

Return the underlying free module corresponding to this space of modular forms.

**EXAMPLES:**

```python
sage: m = ModularForms(Gamma1(13),10)
sage: m.free_module()
Vector space of dimension 69 over Rational Field
sage: ModularForms(Gamma1(13),4, GF(49,'b')).free_module()
Vector space of dimension 27 over Finite Field in b of size 7^2
```

**new_submodule(p=None)**

Return the new or \( p \)-new submodule of this ambient module.

**INPUT:**

- \( p \) - (default: None), if specified return only the \( p \)-new submodule.

**EXAMPLES:**

```python
sage: m = ModularForms(Gamma0(33),2); m
Modular Forms space of dimension 6 for Congruence Subgroup Gamma0(33) of weight 2 over Rational Field
sage: m.new_submodule()
Modular Forms subspace of dimension 1 of Modular Forms space of dimension 6 for Congruence Subgroup Gamma0(33) of weight 2 over Rational Field

Another example:

```python
sage: M = ModularForms(17,4)
sage: N = M.new_subspace(); N
Modular Forms subspace of dimension 4 of Modular Forms space of dimension 6 for Congruence Subgroup Gamma0(17) of weight 4 over Rational Field
sage: N.basis()
[ q + 2*q^5 + O(q^6),
  q^2 - 3/2*q^5 + O(q^6),
  q^3 + O(q^6),
  q^4 - 1/2*q^5 + O(q^6) ]
```
```python
sage: ModularForms(12,4).new_submodule()
Modular Forms subspace of dimension 1 of Modular Forms space of dimension 9 for Congruence Subgroup Gamma0(12) of weight 4 over Rational Field
```

Unfortunately (TODO) - \( p \)-new submodules aren't yet implemented:

```python
sage: m.new_submodule(3)  # not implemented
Traceback (most recent call last):
... NotImplementedError
sage: m.new_submodule(11)  # not implemented
Traceback (most recent call last):
... NotImplementedError
```

**prec**\((new\_prec=None)\)

Set or get default initial precision for printing modular forms.

**INPUT:**

- \( new\_prec \) - positive integer (default: None)

**OUTPUT:** if \( new\_prec \) is None, returns the current precision.

**EXAMPLES:**

```python
sage: M = ModularForms(1,12, prec=3)
sage: M.prec()
3
sage: M.basis()
[q - 24*q^2 + O(q^3),
 1 + 65520/691*q + 134250480/691*q^2 + O(q^3)]
```

```python
sage: M.prec(5)
5
sage: M.basis()
[q - 24*q^2 + 252*q^3 - 1472*q^4 + O(q^5),
 1 + 65520/691*q + 134250480/691*q^2 + 11606736960/691*q^3 + 274945048560/691*q^4 + O(q^5)]
```

**rank()**

This is a synonym for \( self\.dimension() \).

**EXAMPLES:**

```python
sage: m = ModularForms(Gamma0(20),4)
sage: m.rank()
12
sage: m.dimension()
12
```

1.3. Ambient spaces of modular forms  27
set_precision\((n)\)

Set the default precision for displaying elements of this space.

EXAMPLES:

```
sage: m = ModularForms(Gamma1(5),2)
sage: m.set_precision(10)
sage: m.basis()
[1 + 60*q^3 - 120*q^4 + 240*q^5 - 300*q^6 + 300*q^7 - 180*q^9 + O(q^10),
 q + 6*q^3 - 9*q^4 + 27*q^5 - 28*q^6 + 30*q^7 - 11*q^9 + O(q^10),
 q^2 - 4*q^3 + 12*q^4 - 22*q^5 + 30*q^6 - 24*q^7 + 5*q^8 + 18*q^9 + O(q^10)]
sage: m.set_precision(5)
sage: m.basis()
[1 + 60*q^3 - 120*q^4 + O(q^5),
 q + 6*q^3 - 9*q^4 + O(q^5),
 q^2 - 4*q^3 + 12*q^4 + O(q^5)]
```

1.4 Modular forms with character

EXAMPLES:

```
sage: eps = DirichletGroup(13).0
sage: M = ModularForms(eps^2, 2); M
Modular Forms space of dimension 3, character \([\zeta_6]\) and weight 2 over Cyclotomic Field \(\rightarrow\) of order 6 and degree 2
sage: S = M.cuspidal_submodule(); S
Cuspidal subspace of dimension 1 of Modular Forms space of dimension 3, character \(\rightarrow\) \([\zeta_6]\) and weight 2 over Cyclotomic Field of order 6 and degree 2
sage: S.modular_symbols()
Modular Symbols subspace of dimension 2 of Modular Symbols space of dimension 4 and \(\rightarrow\) level 13, weight 2, character \([\zeta_6]\), sign 0, over Cyclotomic Field of order 6 and \(\rightarrow\) degree 2
```

We create a spaces associated to Dirichlet characters of modulus 225:

```
sage: e = DirichletGroup(225).0
sage: e.order()
6
sage: e.base_ring()
Cyclotomic Field of order 60 and degree 16
sage: M = ModularForms(e,3)
```

Notice that the base ring is “minimized”:

```
sage: M
Modular Forms space of dimension 66, character \([\zeta_6, 1]\) and weight 3 over Cyclotomic Field of order 6 and degree 2
```
If we don’t want the base ring to change, we can explicitly specify it:

```
sage: ModularForms(e, 3, e.base_ring())
Modular Forms space of dimension 66, character [zeta6, 1] and weight 3
over Cyclotomic Field of order 60 and degree 16
```

Next we create a space associated to a Dirichlet character of order 20:

```
sage: e = DirichletGroup(225).1
sage: e.order()
20
sage: e.base_ring()
Cyclotomic Field of order 60 and degree 16
sage: M = ModularForms(e, 17); M
Modular Forms space of dimension 484, character [1, zeta20] and
weight 17 over Cyclotomic Field of order 20 and degree 8
```

We compute the Eisenstein subspace, which is fast even though the dimension of the space is large (since an explicit basis of $q$-expansions has not been computed yet).

```
sage: M.eisenstein_submodule()
Eisenstein subspace of dimension 8 of Modular Forms space of
dimension 484, character [1, zeta20] and weight 17 over Cyclotomic Field of order 20 and
degree 8
sage: M.cuspidal_submodule()
Cuspidal subspace of dimension 476 of Modular Forms space of dimension 484, character [1,
 zeta20] and weight 17 over Cyclotomic Field of order 20 and degree 8
```

1.4. Modular forms with character
cuspidal submodule()

Return the cuspidal submodule of this ambient space of modular forms.

EXAMPLES:

```
sage: eps = DirichletGroup(4).0
sage: M = ModularForms(eps, 5); M
Modular Forms space of dimension 3, character [-1] and weight 5 over Rational Field
sage: M.cuspidal_submodule()
Cuspidal subspace of dimension 1 of Modular Forms space of dimension 3, character [-1] and weight 5 over Rational Field
```

eisenstein submodule()

Return the submodule of this ambient module with character that is spanned by Eisenstein series. This is the Hecke stable complement of the cuspidal submodule.

EXAMPLES:

```
sage: m = ModularForms(DirichletGroup(13).0^2,2); m
Modular Forms space of dimension 3, character [zeta6] and weight 2 over Cyclotomic Field of order 6 and degree 2
sage: m.eisenstein_submodule()
Eisenstein subspace of dimension 2 of Modular Forms space of dimension 3, character [zeta6] and weight 2 over Cyclotomic Field of order 6 and degree 2
```

hecke_module_of_level(N)

Return the Hecke module of level N corresponding to self, which is the domain or codomain of a degeneracy map from self. Here N must be either a divisor or a multiple of the level of self, and a multiple of the conductor of the character of self.

EXAMPLES:

```
sage: M = ModularForms(DirichletGroup(15).0, 3); M.character().conductor()
3
sage: M.hecke_module_of_level(3)
Modular Forms space of dimension 2, character [-1] and weight 3 over Rational Field
sage: M.hecke_module_of_level(5)
Traceback (most recent call last):
  ... ValueError: conductor(=3) must divide M(=5)
sage: M.hecke_module_of_level(30)
Modular Forms space of dimension 16, character [-1, 1] and weight 3 over Rational Field
```

modular symbols(sign=0)

Return corresponding space of modular symbols with given sign.

EXAMPLES:

```
sage: eps = DirichletGroup(13).0
sage: N = ModularForms(eps^2, 2)
sage: N.modular_symbols()
Modular Symbols space of dimension 4 and level 13, weight 2, character [zeta6],...
```
→ sign 0, over Cyclotomic Field of order 6 and degree 2
sage: M.modular_symbols(1)
Modular Symbols space of dimension 3 and level 13, weight 2, character [zeta6],
→ sign 1, over Cyclotomic Field of order 6 and degree 2
sage: M.modular_symbols(-1)
Modular Symbols space of dimension 1 and level 13, weight 2, character [zeta6],
→ sign -1, over Cyclotomic Field of order 6 and degree 2
sage: M.modular_symbols(2)
Traceback (most recent call last):
...
ValueError: sign must be -1, 0, or 1

1.5 Modular forms for $\Gamma_0(N)$ over $\mathbb{Q}$

class sage.modular.modform.ambient_g0.ModularFormsAmbient_g0_Q(level, weight)
  Bases: ModularFormsAmbient
  A space of modular forms for $\Gamma_0(N)$ over $\mathbb{Q}$.

cuspidal_submodule()
  Return the cuspidal submodule of this space of modular forms for $\Gamma_0(N)$.

EXAMPLES:

sage: m = ModularForms(Gamma0(33),4)
sage: s = m.cuspidal_submodule(); s
Cuspidal subspace of dimension 10 of Modular Forms space of dimension 14 for $\mathbb{Q}$
→ Congruence Subgroup Gamma0(33) of weight 4 over Rational Field
sage: type(s)
<class 'sage.modular.modform.cuspidal_submodule.CuspidalSubmodule_g0_Q_with_
→ category'>

eisenstein_submodule()
  Return the Eisenstein submodule of this space of modular forms for $\Gamma_0(N)$.

EXAMPLES:

sage: m = ModularForms(Gamma0(389),6)
sage: m.eisenstein_submodule()
Eisenstein subspace of dimension 2 of Modular Forms space of dimension 163 for $\mathbb{Q}$
→ Congruence Subgroup Gamma0(389) of weight 6 over Rational Field
### 1.6 Modular forms for $\Gamma_1(N)$ and $\Gamma_H(N)$ over $\mathbb{Q}$

EXAMPLES:

```python
sage: M = ModularForms(Gamma1(13),2); M
Modular Forms space of dimension 13 for Congruence Subgroup Gamma1(13) of weight 2 over Rational Field
sage: S = M.cuspidal_submodule(); S
Cuspidal subspace of dimension 2 of Modular Forms space of dimension 13 for Congruence Subgroup Gamma1(13) of weight 2 over Rational Field
sage: S.basis()
[ q - 4*q^3 - q^4 + 3*q^5 + O(q^6),
  q^2 - 2*q^3 - q^4 + 2*q^5 + O(q^6) ]
sage: M = ModularForms(GammaH(11, [3])); M
Modular Forms space of dimension 2 for Congruence Subgroup Gamma_H(11) with H generated by [3] of weight 2 over Rational Field
sage: M.q_expansion_basis(8)
[ q - 2*q^2 - q^3 + 2*q^4 + q^5 + 2*q^6 - 2*q^7 + O(q^8),
  1 + 12/5*q + 36/5*q^2 + 48/5*q^3 + 84/5*q^4 + 72/5*q^5 + 144/5*q^6 + 96/5*q^7 + O(q^8) ]
```

class `sage.modular.modform.ambient_g1.ModularFormsAmbient_g1_Q(level, weight, eis_only)`
Bases: `ModularFormsAmbient_gH_Q`

A space of modular forms for the group $\Gamma_1(N)$ over the rational numbers.

cuspidal_submodule()

Return the cuspidal submodule of this modular forms space.

EXAMPLES:

```python
sage: m = ModularForms(Gamma1(17),2); m
Modular Forms space of dimension 20 for Congruence Subgroup Gamma1(17) of weight 2 over Rational Field
sage: m.cuspidal_submodule()
Cuspidal subspace of dimension 5 of Modular Forms space of dimension 20 for Congruence Subgroup Gamma1(17) of weight 2 over Rational Field
```

eisenstein_submodule()

Return the Eisenstein submodule of this modular forms space.

EXAMPLES:

```python
sage: ModularForms(Gamma1(13),2).eisenstein_submodule()
Eisenstein subspace of dimension 11 of Modular Forms space of dimension 13 for Congruence Subgroup Gamma1(13) of weight 2 over Rational Field
sage: ModularForms(Gamma1(13),10).eisenstein_submodule()
Eisenstein subspace of dimension 12 of Modular Forms space of dimension 69 for Congruence Subgroup Gamma1(13) of weight 10 over Rational Field
```
class sage.modular.modform.ambient_g1.ModularFormsAmbient_gH_Q(group, weight, eis_only)

Bases: ModularFormsAmbient

A space of modular forms for the group $\Gamma_H(N)$ over the rational numbers.

cuspidal submodule()

Return the cuspidal submodule of this modular forms space.

EXAMPLES:

```sage
sage: m = ModularForms(GammaH(100, [29]),2); m
Modular Forms space of dimension 48 for Congruence Subgroup Gamma_H(100) with H generated by [29] of weight 2 over Rational Field
sage: m.cuspidal_submodule()
Cuspidal subspace of dimension 13 of Modular Forms space of dimension 48 for Congruence Subgroup Gamma_H(100) with H generated by [29] of weight 2 over Rational Field
```

eisenstein submodule()

Return the Eisenstein submodule of this modular forms space.

EXAMPLES:

```sage
sage: E = ModularForms(GammaH(100, [29]),3).eisenstein_submodule(); E
Eisenstein subspace of dimension 24 of Modular Forms space of dimension 72 for Congruence Subgroup Gamma_H(100) with H generated by [29] of weight 3 over Rational Field
sage: type(E)
<class 'sage.modular.modform.eisenstein_submodule.EisensteinSubmodule_gH_Q_with_category'>
```

1.7 Modular forms over a non-minimal base ring

class sage.modular.modform.ambient_R.ModularFormsAmbient_R(M, base_ring)

Bases: ModularFormsAmbient

Ambient space of modular forms over a ring other than QQ.

EXAMPLES:

```sage
sage: M = ModularForms(23,2,base_ring=GF(7)) # indirect doctest
sage: M
Modular Forms space of dimension 3 for Congruence Subgroup Gamma0(23) of weight 2 over Finite Field of size 7
sage: M == loads(dumps(M))
True
```

change_ring($R$)

Return this modular forms space with the base ring changed to the ring $R$.

EXAMPLES:
```python
sage: chi = DirichletGroup(109, CyclotomicField(3)).0
sage: M9 = ModularForms(chi, 2, base_ring = CyclotomicField(9))
sage: M9.change_ring(CyclotomicField(15))
Modular Forms space of dimension 10, character [zeta3 + 1] and weight 2 over
Cyclotomic Field of order 15 and degree 8
sage: M9.change_ring(QQ)
Traceback (most recent call last):
... 
ValueError: Space cannot be defined over Rational Field
```

**cuspidal_submodule()**

Return the cuspidal subspace of this space.

**EXAMPLES:**

```python
sage: C = CuspForms(7, 4, base_ring=CyclotomicField(5)) # indirect doctest
sage: type(C)
<class 'sage.modular.modform.cuspidal_submodule.CuspidalSubmodule_R_with_
category'>
```

**modular_symbols**(sign=0)

Return the space of modular symbols attached to this space, with the given sign (default 0).

### 1.8 Submodules of spaces of modular forms

**EXAMPLES:**

```python
sage: M = ModularForms(Gamma1(13),2); M
Modular Forms space of dimension 13 for Congruence Subgroup Gamma1(13) of weight 2 over
Rational Field
sage: M.eisenstein_subspace()
Eisenstein subspace of dimension 11 of Modular Forms space of dimension 13 for
Congruence Subgroup Gamma1(13) of weight 2 over Rational Field
sage: M == loads(dumps(M))
True
sage: M.cuspidal_subspace()
Cuspidal subspace of dimension 2 of Modular Forms space of dimension 13 for Congruence_
Subgroup Gamma1(13) of weight 2 over Rational Field
```

**class** `sage.modular.modform.submodule.ModularFormsSubmodule`(ambient_module, submodule, dual=None, check=False)

Bases: `ModularFormsSpace, HeckeSubmodule`

A submodule of an ambient space of modular forms.

**class** `sage.modular.modform.submodule.ModularFormsSubmoduleWithBasis`(ambient_module, submodule, dual=None, check=False)

Bases: `ModularFormsSubmodule`
1.9 The cuspidal subspace

EXAMPLES:

```python
sage: S = CuspForms(SL2Z,12); S
Cuspidal subspace of dimension 1 of Modular Forms space of dimension 2 for
Modular Group SL(2,Z) of weight 12 over Rational Field
sage: S.basis()
[ q - 24*q^2 + 252*q^3 - 1472*q^4 + 4830*q^5 + O(q^6) ]
```

```python
sage: S = CuspForms(Gamma0(33),2); S
Cuspidal subspace of dimension 3 of Modular Forms space of dimension 6 for
Congruence Subgroup Gamma0(33) of weight 2 over Rational Field
sage: S.basis()
[ q - q^5 + O(q^6),
  q^2 - q^4 - q^5 + O(q^6),
  q^3 + O(q^6) ]
```

```python
sage: S = CuspForms(Gamma1(3),6); S
Cuspidal subspace of dimension 1 of Modular Forms space of dimension 3 for
Congruence Subgroup Gamma1(3) of weight 6 over Rational Field
sage: S.basis()
[ q - 6*q^2 + 9*q^3 + 4*q^4 + 6*q^5 + O(q^6) ]
```

```python
class sage.modular.modform.cuspidal_submodule.CuspidalSubmodule(ambient_space)

Bases: ModularFormsSubmodule

Base class for cuspidal submodules of ambient spaces of modular forms.

change_ring(R)

Change the base ring of self to R, when this makes sense.

This differs from base_extend() in that there may not be a canonical map from self to the new space, as in the first example below. If this space has a character then this may fail when the character cannot be defined over R, as in the second example.

EXAMPLES:

```python
sage: chi = DirichletGroup(109, CyclotomicField(3)).0
sage: S9 = CuspForms(chi, 2, base_ring = CyclotomicField(9)); S9
Cuspidal subspace of dimension 8 of Modular Forms space of dimension 10,␣
˓→character [zeta3 + 1] and weight 2 over Cyclotomic Field of order 9 and␣
˓→degree 6
sage: S9.change_ring(CyclotomicField(3))
Cuspidal subspace of dimension 8 of Modular Forms space of dimension 10,␣
˓→character [zeta3 + 1] and weight 2 over Cyclotomic Field of order 3 and␣
˓→degree 2
sage: S9.change_ring(QQ)
```

(continues on next page)
Traceback (most recent call last):
...
ValueError: Space cannot be defined over Rational Field

**is_cuspidal()**

Return True since spaces of cusp forms are cuspidal.

**EXAMPLES:**

```python
sage: CuspForms(4,10).is_cuspidal()
True
```

**modular_symbols**(sign=0)

Return the corresponding space of modular symbols with the given sign.

**EXAMPLES:**

```python
sage: S = ModularForms(11,2).cuspidal_submodule()
sage: S.modular_symbols()
Modular Symbols subspace of dimension 2 of Modular Symbols space of dimension 3 for Gamma_0(11) of weight 2 with sign 0 over Rational Field
sage: S.modular_symbols(sign=-1)
Modular Symbols subspace of dimension 1 of Modular Symbols space of dimension 1 for Gamma_0(11) of weight 2 with sign -1 over Rational Field
sage: M = S.modular_symbols(sign=1); M
Modular Symbols subspace of dimension 1 of Modular Symbols space of dimension 2 for Gamma_0(11) of weight 2 with sign 1 over Rational Field
sage: M.sign()
1
sage: S = ModularForms(1,12).cuspidal_submodule()
sage: S.modular_symbols()
Modular Symbols subspace of dimension 2 of Modular Symbols space of dimension 3 for Gamma_0(1)(1) of weight 12 with sign 0 over Rational Field
sage: eps = DirichletGroup(13).0
sage: S = CuspForms(eps^2, 2)
S.modular_symbols(sign=0)
Modular Symbols subspace of dimension 2 of Modular Symbols space of dimension 4 and level 13, weight 2, character [zeta6], sign 0, over Cyclotomic Field of order 6 and degree 2
sage: S.modular_symbols(sign=1)
Modular Symbols subspace of dimension 3 and level 13, weight 2, character [zeta6], sign 1, over Cyclotomic Field of order 6 and degree 2
sage: S.modular_symbols(sign=-1)
Modular Symbols subspace of dimension 1 and level 13, weight 2, character [zeta6], sign -1, over Cyclotomic Field of order 6 and degree 2
```
class sage.modular.modform.cuspidal_submodule.CuspidalSubmodule_R(ambient_space)
    Bases: CuspidalSubmodule
    Cuspidal submodule over a non-minimal base ring.

class sage.modular.modform.cuspidal_submodule.CuspidalSubmodule_eps(ambient_space)
    Bases: CuspidalSubmodule_modsym_qexp
    Space of cusp forms with given Dirichlet character.

    EXAMPLES:

    sage: S = CuspForms(DirichletGroup(5).0,5); S
    Cuspidal subspace of dimension 1 of Modular Forms space of dimension 3, character
    \([\zeta_4]\) and weight 5 over Cyclotomic Field of order 4 and degree 2
    sage: S.basis()
    [q + (-zeta4 - 1)*q^2 + (6*zeta4 - 6)*q^3 - 14*zeta4*q^4 + (15*zeta4 + 20)*q^5 + O(q^6)]
    sage: f = S.0
    sage: f.qexp()
    q + (-zeta4 - 1)*q^2 + (6*zeta4 - 6)*q^3 - 14*zeta4*q^4 + (15*zeta4 + 20)*q^5 + O(q^6)
    sage: f.qexp(7)
    q + (-zeta4 - 1)*q^2 + (6*zeta4 - 6)*q^3 - 14*zeta4*q^4 + (15*zeta4 + 20)*q^5 + 12*q^6 + O(q^7)
    sage: f.qexp(3)
    q + (-zeta4 - 1)*q^2 + O(q^3)
    sage: f.qexp(2)
    q + O(q^2)
    sage: f.qexp(1)
    O(q^1)

class sage.modular.modform.cuspidal_submodule.CuspidalSubmodule_g0_Q(ambient_space)
    Bases: CuspidalSubmodule_modsym_qexp
    Space of cusp forms for \(\Gamma_0(N)\) over \(\mathbb{Q}\).

class sage.modular.modform.cuspidal_submodule.CuspidalSubmodule_g1_Q(ambient_space)
    Bases: CuspidalSubmodule_gH_Q
    Space of cusp forms for \(\Gamma_1(N)\) over \(\mathbb{Q}\).

class sage.modular.modform.cuspidal_submodule.CuspidalSubmodule_gH_Q(ambient_space)
    Bases: CuspidalSubmodule_modsym_qexp
    Space of cusp forms for \(\Gamma_H(N)\) over \(\mathbb{Q}\).

class sage.modular.modform.cuspidal_submodule.CuspidalSubmodule_level1_Q(ambient_space)
    Bases: CuspidalSubmodule
    Space of cusp forms of level 1 over \(\mathbb{Q}\).
class sage.modular.modform.cuspidal_submodule.CuspidalSubmodule_modsym_qexp(ambient_space)
    Bases: CuspidalSubmodule

    Cuspidal submodule with \(q\)-expansions calculated via modular symbols.

    hecke_polynomial\( (n, \text{var}=\textit{x}^\prime) \)
    Return the characteristic polynomial of the Hecke operator \(T_n\) on this space. This is computed via modular symbols, and in particular is faster to compute than the matrix itself.

    EXAMPLES:

    sage: CuspForms(105, 2).hecke_polynomial(2, \text{'y'})
    y^13 + 5*y^12 - 4*y^11 - 52*y^10 - 34*y^9 + 174*y^8 + 212*y^7 - 196*y^6 - 375*y^\prime
    - 5 - 11*y^4 + 200*y^3 + 80*y^2

    Check that this gives the same answer as computing the Hecke matrix:

    sage: CuspForms(105, 2).hecke_matrix(2).charpoly(var='y')
    y^13 + 5*y^12 - 4*y^11 - 52*y^10 - 34*y^9 + 174*y^8 + 212*y^7 - 196*y^6 - 375*y^\prime
    - 5 - 11*y^4 + 200*y^3 + 80*y^2

    Check that github issue #21546 is fixed (this example used to take about 5 hours):

    sage: CuspForms(1728, 2).hecke_polynomial(2) # long time (20 sec)
    x^253 + x^251 - 2*x^249

    new_submodule\( (p=None) \)
    Return the new subspace of this space of cusp forms. This is computed using modular symbols.

    EXAMPLES:

    sage: CuspForms(55).new_submodule()
    Modular Forms subspace of dimension 3 of Modular Forms space of dimension 8 for Congruence Subgroup Gamma0(55) of weight 2 over Rational Field

class sage.modular.modform.cuspidal_submodule.CuspidalSubmodule_wt1_eps(ambient_space)
    Bases: CuspidalSubmodule

    Space of cusp forms of weight 1 with specified character.

class sage.modular.modform.cuspidal_submodule.CuspidalSubmodule_wt1_gH(ambient_space)
    Bases: CuspidalSubmodule

    Space of cusp forms of weight 1 for a GammaH group.

1.10 The Eisenstein subspace

class sage.modular.modform.eisenstein_submodule.EisensteinSubmodule(ambient_space)
    Bases: ModularFormsSubmodule

    The Eisenstein submodule of an ambient space of modular forms.

eisenstein_submodule()
    Return the Eisenstein submodule of self. (Yes, this is just self.)

    EXAMPLES:
```python
sage: E = ModularForms(23,4).eisenstein_subspace()
sage: E == E.eisenstein_submodule()
True
```

### modular_symbols\((\text{sign}=0)\)

Return the corresponding space of modular symbols with given sign. This will fail in weight 1.

**Warning:** If sign \(!= 0\), then the space of modular symbols will, in general, only correspond to a *subspace* of this space of modular forms. This can be the case for both sign +1 or -1.

**EXAMPLES:**

```python
sage: E = ModularForms(11,2).eisenstein_submodule()
sage: M = E.modular_symbols(); M
Modular Symbols subspace of dimension 1 of Modular Symbols space of dimension 3 for \Gamma_0(11) of weight 2 with sign 0 over Rational Field
sage: M.sign()
0
sage: M = E.modular_symbols(sign=-1); M
Modular Symbols subspace of dimension 0 of Modular Symbols space of dimension 1 for \Gamma_0(11) of weight 2 with sign -1 over Rational Field
sage: E = ModularForms(1,12).eisenstein_submodule()
sage: E.modular_symbols()
Modular Symbols subspace of dimension 1 of Modular Symbols space of dimension 3 for \Gamma_0(1) of weight 12 with sign 0 over Rational Field
sage: eps = DirichletGroup(13).0
sage: E = EisensteinForms(eps^2, 2)
sage: E.modular_symbols()
Modular Symbols subspace of dimension 2 of Modular Symbols space of dimension 4 and level 13, weight 2, character \([\zeta6]\), sign 0, over Cyclotomic Field of order 6 and degree 2
sage: E = EisensteinForms(eps, 1); E
Eisenstein subspace of dimension 1 of Modular Forms space of character \([\zeta12]\) and weight 1 over Cyclotomic Field of order 12 and degree 4
sage: E.modular_symbols()
Traceback (most recent call last):
 ... ValueError: the weight must be at least 2
```

```python
class sage.modular.modform.eisenstein_submodule.EisensteinSubmodule_eps(ambient_space)
```

Bases: EisensteinSubmodule_params

Space of Eisenstein forms with given Dirichlet character.

**EXAMPLES:**

```python
sage: e = DirichletGroup(27,CyclotomicField(3)).0**2
sage: M = ModularForms(e,2,prec=10).eisenstein_subspace()
(continues on next page)
```
sage: M.dimension()
6

sage: M.eisenstein_series()

\[ -\frac{1}{3}\zeta_6 - \frac{1}{3} + q + (2^zeta_6 - 1)^{q^2} + q^3 + (-2^zeta_6 - 1)^{q^4} + (-5^zeta_6 + 1)^{q^5} + 0(q^6), \]
\[ -\frac{1}{3}\zeta_6 - \frac{1}{3} + q^3 + 0(q^6), \]
\[ q + (-2^zeta_6 + 1)^{q^2} + (-2^zeta_6 - 1)^{q^4} + 5^zeta_6 - 1)^{q^5} + 0(q^6), \]
\[ q + (zeta_6 + 1)^{q^2} + 3^q^3 + (zeta_6 + 2)^{q^4} + (-zeta_6 + 5)^{q^5} + 0(q^6), \]
\[ q^3 + 0(q^6), \]
\[ q + (-zeta_6 - 1)^{q^2} + (zeta_6 + 2)^{q^4} + (-zeta_6 - 5)^{q^5} + 0(q^6) \]

sage: M.eisenstein_subspace().T(2).matrix().fcp()

\[(x + 2\zeta_3 + 1) * (x + \zeta_3 + 2) * (x - \zeta_3 - 2)^2 * (x - 2\zeta_3 - 1)^2\]

sage: ModularSymbols(e,2).eisenstein_subspace().T(2).matrix().fcp()

\[(x + 2\zeta_3 + 1) * (x + \zeta_3 + 2) * (x - \zeta_3 - 2)^2 * (x - 2\zeta_3 - 1)^2\]

sage: M.basis()

\[
1 - 3\zeta_3 q^6 + (-2^zeta_3 + 2)^q^9 + 0(q^10),
q + (5^zeta_3 + 5)^q^7 + 0(q^10),
q^2 - 2^zeta_3 q^8 + 0(q^10),
q^3 + (zeta_3 + 2)^q^6 + 3^q^9 + 0(q^10),
q^4 - 2^zeta_3 q^7 + 0(q^10),
q^5 + (zeta_3 + 1)^q^8 + 0(q^10)
\]

class sage.modular.modform.eisenstein_submodule.EisensteinSubmodule_g0_Q(ambient_space)

Bases: EisensteinSubmodule_params

Space of Eisenstein forms for \(\Gamma_0(N)\).

class sage.modular.modform.eisenstein_submodule.EisensteinSubmodule_g1_Q(ambient_space)

Bases: EisensteinSubmodule_gH_Q

Space of Eisenstein forms for \(\Gamma_1(N)\).

class sage.modular.modform.eisenstein_submodule.EisensteinSubmodule_gH_Q(ambient_space)

Bases: EisensteinSubmodule_params

Space of Eisenstein forms for \(\Gamma_H(N)\).

class sage.modular.modform.eisenstein_submodule.EisensteinSubmodule_params(ambient_space)

change_ring(base_ring)

Return self as a module over base_ring.

EXAMPLES:

sage: E = EisensteinForms(12,2) ; E
Eisenstein subspace of dimension 5 of Modular Forms space of dimension 5 for \Gamma_0(12) of weight 2 over Rational Field
sage: E.basis()
Eisenstein subspace of dimension 5 of Modular Forms space of dimension 5 for Congruence Subgroup Gamma0(12) of weight 2 over Finite Field of size 5

sage: E.change_ring(GF(5)).basis()

\[
\begin{align*}
1 & + O(q^6), \\
q & + 6q^5 + 0(q^6), \\
q^2 & + 0(q^6), \\
q^3 & + 0(q^6), \\
q^4 & + 0(q^6)
\end{align*}
\]

eisenstein_series()

Return the Eisenstein series that span this space (over the algebraic closure).

EXAMPLES:

sage: EisensteinForms(11,2).eisenstein_series()

\[
\begin{align*}
5/12 & + q + 3q^2 + 4q^3 + 7q^4 + 6q^5 + O(q^6)
\end{align*}
\]

sage: EisensteinForms(1,4).eisenstein_series()

\[
\begin{align*}
1/240 & + q + 9q^2 + 28q^3 + 73q^4 + 126q^5 + O(q^6)
\end{align*}
\]

sage: EisensteinForms(1,24).eisenstein_series()

\[
\begin{align*}
236364091/131040 + q + 8388609q^2 + 94143178828q^3 + 70368752566273q^4 + 11920928955078126q^5 + O(q^6)
\end{align*}
\]

sage: EisensteinForms(5,4).eisenstein_series()

\[
\begin{align*}
1/240 & + q + 9q^2 + 28q^3 + 73q^4 + 126q^5 + O(q^6), \\
1/240 & + q^5 + O(q^6)
\end{align*}
\]

sage: EisensteinForms(13,2).eisenstein_series()

\[
\begin{align*}
1/2 & + q + 3q^2 + 4q^3 + 7q^4 + 6q^5 + O(q^6)
\end{align*}
\]

sage: E = EisensteinForms(Gamma1(7),2)
sage: E.set_precision(4)
sage: E.eisenstein_series()

\[
\begin{align*}
1/4 & + q + 3q^2 + 4q^3 + 0(q^4),
\end{align*}
\]

(continues on next page)
1/7*zeta6 - 3/7 + q + (-2*zeta6 + 1)*q^2 + (3*zeta6 - 2)*q^3 + O(q^4),
q + (-zeta6 + 2)*q^2 + (zeta6 + 2)*q^3 + O(q^4),
-1/7*zeta6 - 2/7 + q + (2*zeta6 - 1)*q^2 + (-3*zeta6 + 1)*q^3 + O(q^4),
q + (zeta6 + 1)*q^2 + (-zeta6 + 3)*q^3 + O(q^4)
]
sage: eps = DirichletGroup(13).0^2
sage: ModularForms(eps,2).eisenstein_series()
[[-7/13*zeta6 - 11/13 + q + (2*zeta6 + 1)*q^2 + (-3*zeta6 + 1)*q^3 + (6*zeta6 + 
-3)*q^4 - 4*q^5 + O(q^6),
q + (zeta6 + 2)*q^2 + (-zeta6 + 3)*q^3 + (3*zeta6 + 3)*q^4 + 4*q^5 + O(q^6)]]
sage: M = ModularForms(19,3).eisenstein_subspace()
sage: M.eisenstein_series()
[ ]
sage: M = ModularForms(DirichletGroup(13).0, 1)
sage: M.eisenstein_series()
[[[-1/13*zeta12^3 + 6/13*zeta12^2 + 4/13*zeta12 + 2/13 + q + (zeta12 + 1)*q^2 + zeta12^2*q^3 + (zeta12^2 + zeta12 + 1)*q^4 + (-zeta12^3 + 1)*q^5 + O(q^6)]]
sage: M = ModularForms(GammaH(15, [4]), 4)
sage: M.eisenstein_series()
[[1/240 + q + 9*q^2 + 28*q^3 + 73*q^4 + 126*q^5 + O(q^6),
1/240 + q^3 + O(q^6),
1/240 + q^5 + O(q^6),
1/240 + O(q^6),
1 + q - 7*q^2 - 26*q^3 + 57*q^4 + q^5 + O(q^6),
q + 7*q^2 + 26*q^3 + 57*q^4 + 125*q^5 + O(q^6),
q^3 + O(q^6)]]

new_eisenstein_series()
Return a list of the Eisenstein series in this space that are new.

EXAMPLES:

sage: E = EisensteinForms(25, 4)
sage: E.new_eisenstein_series()
[[q + 7*zeta4*q^2 - 26*zeta4*q^3 - 57*q^4 + O(q^6),
q - 9*q^2 - 28*q^3 + 73*q^4 + O(q^6),
q - 7*zeta4*q^2 + 26*zeta4*q^3 - 57*q^4 + O(q^6)]]

new_submodule(p=None)
Return the new submodule of self.

EXAMPLES:
sage: e = EisensteinForms(Gamma0(225), 2).new_submodule(); e
Modular Forms subspace of dimension 3 of Modular Forms space of dimension 42
for Congruence Subgroup Gamma0(225) of weight 2 over Rational Field
sage: e.basis()
[q + O(q^6),
q^2 + O(q^6),
q^4 + O(q^6)]

parameters()

Return a list of parameters for each Eisenstein series spanning self. That is, for each such series, return
a triple of the form (\( \psi, \chi, \text{level} \)), where \( \psi \) and \( \chi \) are the characters defining the Eisenstein series, and level is
the smallest level at which this series occurs.

EXAMPLES:

sage: ModularForms(24,2).eisenstein_submodule().parameters()
[(Dirichlet character modulo 24 of conductor 1 mapping 7 |--> 1, 13 |--> 1, 17 |--> 1, 17 |--> 1, 2),
...
Dirichlet character modulo 24 of conductor 1 mapping 7 |--> 1, 13 |--> 1, 17 |--> 1, 24)]
sage: EisensteinForms(12,6).parameters()[-1]
(Dirichlet character modulo 12 of conductor 1 mapping 7 |--> 1, 5 |--> 1, 12)
dsage: pars = ModularForms(DirichletGroup(24).0,3).eisenstein_submodule().parameters()
dsage: [(x[0].values_on_gens(),x[1].values_on_gens(),x[2]) for x in pars]
[((1, 1, 1), (-1, 1, 1), 1),
((1, 1, 1), (-1, 1, 1), 2),
((1, 1, 1), (-1, 1, 1), 3),
((1, 1, 1), (-1, 1, 1), 6),
((-1, 1, 1), (1, 1, 1), 1),
((-1, 1, 1), (1, 1, 1), 2),
((-1, 1, 1), (1, 1, 1), 3),
((-1, 1, 1), (1, 1, 1), 6)]
sage: EisensteinForms(DirichletGroup(24).0,1).parameters()
[(Dirichlet character modulo 24 of conductor 1 mapping 7 |--> 1, 13 |--> 1, 17 |--> 1, 17 |--> 1, 17 |--> 1, 17 |--> 1, 17 |--> 1, 2),
(Dirichlet character modulo 24 of conductor 1 mapping 7 |--> 1, 13 |--> 1, 17 |--> 1, 17 |--> 1, 17 |--> 1, 17 |--> 1, 17 |--> 1, 2),
(Dirichlet character modulo 24 of conductor 1 mapping 7 |--> 1, 13 |--> 1, 17 |--> 1, 17 |--> 1, 17 |--> 1, 17 |--> 1, 17 |--> 1, 2),
(Dirichlet character modulo 24 of conductor 1 mapping 7 |--> 1, 13 |--> 1, 17 |--> 1, 17 |--> 1, 17 |--> 1, 17 |--> 1, 17 |--> 1, 2)]
sage: ModularForms(DirichletGroup(24).0,1).parameters()
[(Dirichlet character modulo 24 of conductor 1 mapping 7 |--> 1, 13 |--> 1, 17 |--> 1, 17 |--> 1, 17 |--> 1, 17 |--> 1, 17 |--> 1, 17 |--> 1, 6)]

sage.modular.modform.eisenstein_submodule.cyclotomic_restriction(L, K)

Given two cyclotomic fields L and K, compute the compositum M of K and L, and return a function and the
index \([M:K]\). The function is a map that acts as follows (here \( M = \mathbb{Q}(\zeta_m) \)): 43
INPUT:

element alpha in L

OUTPUT:

a polynomial \( f(x) \) in \( K[x] \) such that \( f(\zeta_m) = \alpha \), where we view alpha as living in \( M \). (Note that \( \zeta_m \) generates \( M \), not \( L \).)

EXAMPLES:

```
sage: L = CyclotomicField(12) ; N = CyclotomicField(33) ; M = CyclotomicField(132)
sage: z, n = sage.modular.modform.eisenstein_submodule.cyclotomic_restriction(L,N)
sage: n
2
sage: z(L.0)
-zeta33^19*x
sage: z(L.0)(M.0)
zeta132^11
sage: z(L.0^3-L.0+1)
(zeta33^19 + zeta33^8)*x + 1
sage: z(L.0^3-L.0+1)(M.0)
zeta132^33 - zeta132^11 + 1
sage: z(L.0^3-L.0+1)(M.0) - M(L.0^3-L.0+1)
0
```

\texttt{sage.modular.modform.eisenstein_submodule.cyclotomic_restriction_tower}(L, K)

Suppose \( L/K \) is an extension of cyclotomic fields and \( L=\mathbb{Q}(\zeta_m) \). This function computes a map with the following property:

INPUT:

an element alpha in L

OUTPUT:

a polynomial \( f(x) \) in \( K[x] \) such that \( f(\zeta_m) = \alpha \).

EXAMPLES:

```
sage: L = CyclotomicField(12) ; K = CyclotomicField(6)
sage: z = sage.modular.modform.eisenstein_submodule.cyclotomic_restriction_tower(L, K)
sage: z(L.0)
x
sage: z(L.0^2+L.0)
x + zeta6
```

Chapter 1. Modular Forms for Arithmetic Groups
1.11 Eisenstein series

`sage.modular.modform.eis_series.compute_eisenstein_params(character, k)`

Compute and return a list of all parameters \((\chi, \psi, t)\) that define the Eisenstein series with given character and weight \(k\).

Only the parity of \(k\) is relevant (unless \(k = 1\), which is a slightly different case).

If \texttt{character} is an integer \(N\), then the parameters for \(\Gamma_1(N)\) are computed instead. Then the condition is that \(\chi(-1) * \psi(-1) = (-1)^k\).

If \texttt{character} is a list of integers, the parameters for \(\Gamma_H(N)\) are computed, where \(H\) is the subgroup of \((\mathbb{Z}/N\mathbb{Z})^\times\) generated by the integers in the given list.

EXAMPLES:

```python
sage: sage.modular.modform.eis_series.compute_eisenstein_params(DirichletGroup(30)(1), 3)
[]
sage: pars = sage.modular.modform.eis_series.compute_eisenstein_params(DirichletGroup(30)(1), 4)
sage: [(x[0].values_on_gens(), x[1].values_on_gens(), x[2]) for x in pars]
[[((1, 1), (1, 1), 1),
  ((1, 1), (1, 1), 2),
  ((1, 1), (1, 1), 3),
  ((1, 1), (1, 1), 5),
  ((1, 1), (1, 1), 6),
  ((1, 1), (1, 1), 10),
  ((1, 1), (1, 1), 15),
  ((1, 1), (1, 1), 30)]

sage: pars = sage.modular.modform.eis_series.compute_eisenstein_params(15, 1)
sage: [(x[0].values_on_gens(), x[1].values_on_gens(), x[2]) for x in pars]
[[(1, 1), (-1, 1), 1],
  ((1, 1), (-1, 1), 5),
  ((1, 1), (1, zeta4), 1),
  ((1, 1), (1, zeta4), 3),
  ((1, 1), (-1, -1), 1),
  ((1, 1), (1, -zeta4), 1),
  ((1, 1), (1, -zeta4), 3),
  ((-1, 1), (1, -1), 1)]

sage: sage.modular.modform.eis_series.compute_eisenstein_params(DirichletGroup(15).gen(0), 1)
[[(Dirichlet character modulo 15 of conductor 1 mapping 11 |--> 1, 7 |--> 1,),
  (Dirichlet character modulo 15 of conductor 3 mapping 11 |--> -1, 7 |--> 1,),
  (Dirichlet character modulo 15 of conductor 1 mapping 11 |--> 1, 7 |--> 1,),
  (Dirichlet character modulo 15 of conductor 3 mapping 11 |--> -1, 7 |--> 1, 5)]

sage: len(sage.modular.modform.eis_series.compute_eisenstein_params(GammaH(15, [4]), -3))
8
```
sage.modular.modform.eis_series.eisenstein_series_lseries(weight, prec=53,
max_imaginary_part=0,
max_asymp_coeffs=40)

Return the L-series of the weight $2k$ Eisenstein series on $\text{SL}_2(\mathbb{Z})$.

This actually returns an interface to Tim Dokchitser’s program for computing with the L-series of the Eisenstein series.

INPUT:

- **weight** - even integer
- **prec** - integer (bits precision)
- **max_imaginary_part** - real number
- **max_asymp_coeffs** - integer

OUTPUT:

The L-series of the Eisenstein series.

EXAMPLES:

We compute with the L-series of $E_{16}$ and then $E_{20}$:

```
sage: L = eisenstein_series_lseries(16)
sage: L(1)
-0.291657724743874
sage: L = eisenstein_series_lseries(20)
sage: L(2)
-5.02355351645998
```

Now with higher precision:

```
sage: L = eisenstein_series_lseries(20, prec=200)
sage: L(2)
-5.023553516459979747196848418348135050804419155747868718371029
```

sage.modular.modform.eis_series.eisenstein_series_qexp(k, prec=10, K=Rational Field, var='q',
normalization='linear')

Return the $q$-expansion of the normalized weight $k$ Eisenstein series on $\text{SL}_2(\mathbb{Z})$ to precision prec in the ring $K$.

Three normalizations are available, depending on the parameter normalization; the default normalization is the one for which the linear coefficient is 1.

INPUT:

- **k** - an even positive integer
- **prec** - (default: 10) a nonnegative integer
- **K** - (default: $\mathbb{Q}$) a ring
- **var** - (default: 'q') variable name to use for q-expansion
- **normalization** - (default: 'linear') normalization to use. If this is 'linear', then the series will be normalized so that the linear term is 1. If it is 'constant', the series will be normalized to have constant term 1. If it is 'integral', then the series will be normalized to have integer coefficients and no common factor, and linear term that is positive. Note that 'integral' will work over arbitrary base rings, while 'linear' or 'constant' will fail if the denominator (resp. numerator) of $B_k/2k$ is invertible.

ALGORITHM:
We know \( E_k = \text{constant} + \sum_n \sigma_{k-1}(n)q^n \). So we compute all the \( \sigma_{k-1}(n) \) simultaneously, using the fact that \( \sigma \) is multiplicative.

**EXAMPLES:**

```
sage: eisenstein_series_qexp(2,5)
-1/24 + q + 3*q^2 + 4*q^3 + 7*q^4 + O(q^5)
sage: eisenstein_series_qexp(2,0)
O(q^0)
sage: eisenstein_series_qexp(2,5,GF(7))
2 + q + 3*q^2 + 4*q^3 + O(q^5)
sage: eisenstein_series_qexp(2,5,GF(7),var='T')
2 + T + 3*T^2 + 4*T^3 + O(T^5)
```

We illustrate the use of the normalization parameter:

```
sage: eisenstein_series_qexp(12, 5, normalization='integral')
691 + 65520*q + 134250480*q^2 + 11606736960*q^3 + 274945048560*q^4 + O(q^5)
sage: eisenstein_series_qexp(12, 5, normalization='constant')
1 + 65520/691*q + 134250480/691*q^2 + 11606736960/691*q^3 + 274945048560/691*q^4 + o(1)
˓→O(q^5)
sage: eisenstein_series_qexp(12, 5, normalization='linear')
691/65520 + q + 2049*q^2 + 177148*q^3 + 4196353*q^4 + O(q^5)
sage: eisenstein_series_qexp(12, 50, K=GF(13), normalization="constant")
1 + O(q^50)
```

**AUTHORS:**

- William Stein: original implementation
- Craig Citro (2007-06-01): rewrote for massive speedup
- Martin Raum (2009-08-02): port to cython for speedup
- David Loeffler (2010-04-07): work around an integer overflow when \( k \) is large
- David Loeffler (2012-03-15): add options for alternative normalizations (motivated by github issue #12043)

### 1.12 Eisenstein series, optimized

`sage.modular.modform.eis_series_cython.Ek_ZZ(k, prec=10)`

Return list of prec integer coefficients of the weight \( k \) Eisenstein series of level 1, normalized so the coefficient of \( q \) is 1, except that the 0th coefficient is set to 1 instead of its actual value.

**INPUT:**

- \( k \) – int
- \( \text{prec} \) – int

**OUTPUT:**

- list of Sage Integers.
sage: from sage.modular.modform.eis_series_cython import Ek_ZZ
sage: Ek_ZZ(4,10)
[1, 1, 9, 28, 73, 126, 252, 344, 585, 757]
sage: [sigma(n,3) for n in [1..9]]
[1, 9, 28, 73, 126, 252, 344, 585, 757]
sage: Ek_ZZ(10,10^3) == [1] + [sigma(n,9) for n in range(1,10^3)]
True

sage.modular.modform.eis_series_cython.eisenstein_series_poly(k, prec=10)
Return the q-expansion up to precision prec of the weight k Eisenstein series, as a FLINT Fmpz_poly object, normalised so the coefficients are integers with no common factor.
Used internally by the functions eisenstein_series_qexp() and victor_miller_basis(): see the docstring of the former for further details.

EXAMPLES:

sage: from sage.modular.modform.eis_series_cython import eisenstein_series_poly
sage: eisenstein_series_poly(12, prec=5)
5 691 65520 134250480 11606736960 274945048560

1.13 Elements of modular forms spaces

Class hierarchy:
- ModularForm_abstract
  - Newform
    * ModularFormElement_elliptic_curve
  - ModularFormElement
    * EisensteinSeries
- GradedModularFormElement

AUTHORS:
- David Ayotte (2021-06): GradedModularFormElement class

class sage.modular.modform.element.EisensteinSeries(parent, vector, t, chi, psi)
Bases: ModularFormElement
An Eisenstein series.

EXAMPLES:

sage: E = EisensteinForms(1,12)
sage: E.eisenstein_series()
[ 691/65520 + q + 2049*q^2 + 177148*q^3 + 4196353*q^4 + 48828126*q^5 + O(q^6) ]
sage: E = EisensteinForms(11,2)
sage: E.eisenstein_series()
(continues on next page)
\[
\begin{align*}
\frac{5}{12} + q + 3q^2 + 4q^3 + 7q^4 + 6q^5 + O(q^6) \\
\text{sage: } E = \text{EisensteinForms}(\Gamma_1(7), 2) \\
\text{sage: } E.set\_precision(4) \\
\text{sage: } E.eisenstein\_series() \\
\end{align*}
\]

\[
\begin{align*}
\left[ \frac{1}{4} + q + 3q^2 + 4q^3 + O(q^4), \\
\frac{1}{7}zeta6 - \frac{3}{7} + q + (-2zeta6 + 1)q^2 + (3zeta6 - 2)q^3 + O(q^4), \\
q + (-zeta6 + 2)q^2 + (zeta6 + 2)q^3 + O(q^4), \\
-\frac{1}{7}zeta6 - \frac{2}{7} + q + (2zeta6 - 1)q^2 + (-3zeta6 + 1)q^3 + O(q^4), \\
q + (zeta6 + 1)q^2 + (-zeta6 + 3)q^3 + O(q^4) \\
\right]
\]

\text{L}() 

Return the conductor of self.chi().

\text{EXAMPLES:}

\[
\text{sage: } \text{EisensteinForms}(\text{DirichletGroup}(17).0, 99).\text{eisenstein\_series()}[1].\text{L}() \\
17
\]

\text{M}() 

Return the conductor of self.psi().

\text{EXAMPLES:}

\[
\text{sage: } \text{EisensteinForms}(\text{DirichletGroup}(17).0, 99).\text{eisenstein\_series()}[1].\text{M}() \\
1
\]

\text{character}() 

Return the character associated to self.

\text{EXAMPLES:}

\[
\text{sage: } \text{chi} = \text{DirichletGroup}(7)[4] \\
\text{sage: } E = \text{EisensteinForms}(\text{chi}).\text{eisenstein\_series()} ; E \\
\begin{align*}
\left[ \frac{-1}{7}zeta6 - \frac{2}{7} + q + (2zeta6 - 1)q^2 + (-3zeta6 + 1)q^3 + (2zeta6 - 1)q^4 + (5zeta6 - 4)q^5 + O(q^6), \\
q + (zeta6 + 1)q^2 + (-zeta6 + 3)q^3 + (zeta6 + 2)q^4 + (zeta6 + 4)q^5 + O(q^6) \\
\right] \\
\text{sage: } E[0].\text{character}() == \text{chi} \\
\text{True} \\
\text{sage: } E[1].\text{character}() == \text{chi} \\
\text{True}
\]

\text{chi}() 

Return the parameter chi associated to self.
EXAMPLES:

```
sage: EisensteinForms(DirichletGroup(17).0,99).eisenstein_series()[1].chi()
Dirichlet character modulo 17 of conductor 17 mapping 3 |--> zeta16
```

new_level()

Return level at which self is new.

EXAMPLES:

```
sage: EisensteinForms(DirichletGroup(17).0,99).eisenstein_series()[1].level()
17
sage: EisensteinForms(DirichletGroup(17).0,99).eisenstein_series()[1].new_level()
17
sage: [ [x.level(), x.new_level()] for x in EisensteinForms(DirichletGroup(60).0^2,2).eisenstein_series() ]
[[60, 2], [60, 3], [60, 2], [60, 5], [60, 2], [60, 2], [60, 3], [60, 2], [60, 2], [60, 2], [60, 2]]
```

parameters()

Return chi, psi, and t, which are the defining parameters of self.

EXAMPLES:

```
sage: EisensteinForms(DirichletGroup(17).0,99).eisenstein_series()[1].parameters()
(Dirichlet character modulo 17 of conductor 17 mapping 3 |--> zeta16, Dirichlet character modulo 17 of conductor 1 mapping 3 |--> 1, 1)
```

psi()

Return the parameter psi associated to self.

EXAMPLES:

```
sage: EisensteinForms(DirichletGroup(17).0,99).eisenstein_series()[1].psi()
Dirichlet character modulo 17 of conductor 1 mapping 3 |--> 1
```

t()

Return the parameter t associated to self.

EXAMPLES:

```
sage: EisensteinForms(DirichletGroup(17).0,99).eisenstein_series()[1].t()
1
```

class sage.modular.modform.element.GradedModularFormElement(parent, forms_datum)

Bases: ModuleElement

The element class for ModularFormsRing. A GradedModularFormElement is basically a formal sum of modular forms of different weight: \( f_1 + f_2 + \ldots + f_n \). Note that a GradedModularFormElement is not necessarily a modular form (as it can have mixed weight components).

A GradedModularFormElement should not be constructed directly via this class. Instead, one should use the element constructor of the parent class (ModularFormsRing).

EXAMPLES:
A graded modular form can be initiated via a dictionary or a list:

\[
\begin{align*}
sage: & E4 = \text{ModularForms}(1, 4).0 \\
sage: & M([4:E4, 12:D]) \quad \# \text{ dictionary} \\
& 1 + 241q + 2136q^2 + 6972q^3 + 16048q^4 + 35070q^5 + O(q^6) \\
sage: & M([E4, D]) \quad \# \text{ list} \\
& 1 + 241q + 2136q^2 + 6972q^3 + 16048q^4 + 35070q^5 + O(q^6)
\end{align*}
\]

Also, when adding two modular forms of different weights, a graded modular form element will be created:

\[
\begin{align*}
sage: & (E4 + D).parent() \\
& \text{Ring of Modular Forms for Modular Group SL(2,\mathbb{Z}) over Rational Field} \\
sage: & M([E4, D]) == E4 + D \\
& True
\end{align*}
\]

Graded modular forms elements for congruence subgroups are also supported:

\[
\begin{align*}
sage: & M = \text{ModularFormsRing}(\text{Gamma0}(3)) \\
sage: & f = \text{ModularForms}(\text{Gamma0}(3), 4).0 \\
sage: & g = \text{ModularForms}(\text{Gamma0}(3), 2).0 \\
sage: & M([f, g]) \\
& 2 + 12q + 36q^2 + 252q^3 + 84q^4 + 72q^5 + O(q^6) \\
sage: & M([4:f, 2:g]) \\
& 2 + 12q + 36q^2 + 252q^3 + 84q^4 + 72q^5 + O(q^6)
\end{align*}
\]

derivative(name='E2')

Return the derivative \(q \frac{d}{dq}\) of the given graded form.

Note that this method returns an element of a new parent, that is a quasimodular form. If the form is not homogeneous, then this method sums the derivative of each homogeneous component.

INPUT:

* name (str, default: 'E2') – the name of the weight 2 Eisenstein series generating the graded algebra of quasimodular forms over the ring of modular forms.

OUTPUT: a \texttt{sage.modular.quasimodform.element.QuasiModularFormsElement}

EXAMPLES:

\[
\begin{align*}
sage: & M = \text{ModularFormsRing}(1) \\
sage: & E4 = M.0; E6 = M.1 \\
sage: & dE4 = E4.derivative(); dE4 \\
& 240q + 4320q^2 + 20160q^3 + 70080q^4 + 151200q^5 + O(q^6) \\
sage: & dE4.parent() \\
& \text{Ring of Quasimodular Forms for Modular Group SL(2,\mathbb{Z}) over Rational Field} \\
sage: & dE4.is_modular_form() \\
& False
\end{align*}
\]

group()

Return the group for which \texttt{self} is a modular form.
EXAMPLES:

```sage
M = ModularFormsRing(1)
E4 = M.0
E4.group()
Modular Group SL(2,Z)
M5 = ModularFormsRing(Gamma1(5))
f = M5(ModularForms(Gamma1(5)).0);
f.group()
Congruence Subgroup Gamma1(5)
```

**homogeneous_component**(weight)

Return the homogeneous component of the given graded modular form.

**INPUT:**

- `weight` – An integer corresponding to the weight of the homogeneous component of the given graded modular form.

**EXAMPLES:**

```sage
M = ModularFormsRing(1)
f4 = ModularForms(1, 4).0; f6 = ModularForms(1, 6).0; f8 = ModularForms(1, 8).0
F = M(f4) + M(f6) + M(f8)
F[4] # indirect doctest
1 + 240*q + 2160*q^2 + 6720*q^3 + 17520*q^4 + 30240*q^5 + O(q^6)
F[6] # indirect doctest
1 - 504*q - 16632*q^2 - 122976*q^3 - 532728*q^4 - 1575504*q^5 + O(q^6)
F[8] # indirect doctest
1 + 480*q + 61920*q^2 + 1050240*q^3 + 7926240*q^4 + 37500480*q^5 + O(q^6)
F[10] # indirect doctest
0
F.homogeneous_component(4)
1 + 240*q + 2160*q^2 + 6720*q^3 + 17520*q^4 + 30240*q^5 + O(q^6)
```

**is_homogeneous()**

Return True if the graded modular form is homogeneous, i.e. if it is a modular forms of a certain weight.

An alias of this method is `is_modular_form`

**EXAMPLES:**

```sage
M = ModularFormsRing(1)
E4 = M.0; E6 = M.1;
E4.is_homogeneous()
True
F = E4 + E6 # Not a modular form
F.is_homogeneous()
False
```

**is_modular_form()**

Return True if the graded modular form is homogeneous, i.e. if it is a modular forms of a certain weight.

An alias of this method is `is_modular_form`

**EXAMPLES:**

```sage
```
```python
sage: M = ModularFormsRing(1)
sage: E4 = M.0; E6 = M.1;
sage: E4.is_homogeneous()
True
sage: F = E4 + E6 # Not a modular form
sage: F.is_homogeneous()
False
```

**is_one()**

Return "True" if the graded form is 1 and "False" otherwise

**EXAMPLES:**

```python
sage: M = ModularFormsRing(1)
sage: M(1).is_one()
True
sage: M(2).is_one()
False
sage: E6 = M.0
sage: E6.is_one()
False
```

**is_zero()**

Return "True" if the graded form is 0 and "False" otherwise

**EXAMPLES:**

```python
sage: M = ModularFormsRing(1)
sage: M(0).is_zero()
True
sage: M(1/2).is_zero()
False
sage: E6 = M.1
sage: M(E6).is_zero()
False
```

**q_expansion**(prec=None)

Return the \(q\)-expansion of the graded modular form up to precision \(prec\) (default: 6).

An alias of this method is qexp.

**EXAMPLES:**

```python
sage: M = ModularFormsRing(1)
sage: zer = M(0); zer.q_expansion()
0
sage: M(5/7).q_expansion()
5/7
sage: E4 = M.0; E4
1 + 240*q + 2160*q^2 + 6720*q^3 + 17520*q^4 + 30240*q^5 + O(q^6)
sage: E6 = M.1; E6
1 - 504*q - 16632*q^2 - 122976*q^3 - 532728*q^4 - 1575504*q^5 + O(q^6)
sage: F = E4 + E6; F
2 - 264*q - 14472*q^2 - 116256*q^3 - 515208*q^4 - 1545264*q^5 + O(q^6)
sage: F.q_expansion()
```

(continues on next page)

---

1.13. Elements of modular forms spaces 53
The q-expansion of the graded modular form up to precision prec (default: 6).

An alias of this method is qexp.

EXAMPLES:

```python
sage: M = ModularFormsRing(1)
sage: zer = M(0); zer.q_expansion()
0
sage: M(5/7).q_expansion()
5/7
sage: E4 = M.0; E4
1 + 240*q + 2160*q^2 + 6720*q^3 + 17520*q^4 + 30240*q^5 + O(q^6)
sage: E6 = M.1; E6
1 - 504*q + 16632*q^2 + 107760*q^3 + 335632*q^4 + 645120*q^5 + O(q^6)
sage: F = E4 + E6; F
2 - 264*q - 14472*q^2 - 116256*q^3 - 515208*q^4 - 1545264*q^5 + O(q^6)
sage: F.q_expansion()
2 - 264*q - 14472*q^2 - 116256*q^3 - 515208*q^4 - 1545264*q^5 + O(q^6)
sage: F.q_expansion(10)
2 - 264*q - 14472*q^2 - 116256*q^3 - 515208*q^4 - 1545264*q^5 - 3997728*q^6 -
˓→8388672*q^7 - 16907400*q^8 - 29701992*q^9 + O(q^10)
```

Return the Serre derivative of the given graded modular form.

If self is a modular form of weight \( k \), then the returned modular form will be of weight \( k + 2 \). If the form is not homogeneous, then this method sums the Serre derivative of each homogeneous component.

EXAMPLES:

```python
sage: M = ModularFormsRing(1)
sage: E4 = M.0
sage: E6 = M.1
sage: DE4 = E4.serre_derivative(); DE4
-1/3 + 168*q + 5544*q^2 + 40992*q^3 + 177576*q^4 + 525168*q^5 + O(q^6)
sage: DE4 == (-1/3) * E6
True
sage: DE6 = E6.serre_derivative(); DE6
-1/2 - 240*q - 30960*q^2 - 525120*q^3 - 3963120*q^4 - 18750240*q^5 + O(q^6)
sage: DE6 == (-1/2) * E4^2
True
sage: f = E4 + E6
sage: Df = f.serre_derivative(); Df
-5/6 - 72*q - 25416*q^2 - 484128*q^3 - 3785544*q^4 - 18225072*q^5 + O(q^6)
sage: Df == (-1/3) * E6 + (-1/2) * E4^2
True
```
sage: M(1/2).serre_derivative()
0

to_polynomial(names='x', gens=None)
Return a polynomial $P(x_0, ..., x_n)$ such that $P(g_0, ..., g_n)$ is equal to self where $g_0, ..., g_n$ is a list of generators of the parent.

INPUT:
• names – a list or tuple of names (strings), or a comma separated string. Correspond to the names of the variables;
• gens – (default: None) a list of generator of the parent of self. If set to None, the list returned by gen_forms() is used instead

OUTPUT: A polynomial in the variables names

EXAMPLES:

sage: M = ModularFormsRing(1)
sage: (M.0 + M.1).to_polynomial()
x1 + x0
sage: (M.0^10 + M.0 * M.1).to_polynomial()
x0^10 + x0*x1

This method is not necessarily the inverse of from_polynomial() since there may be some relations between the generators of the modular forms ring:

sage: M = ModularFormsRing(Gamma0(6))
sage: P.<x0,x1,x2> = M.polynomial_ring()
sage: M.from_polynomial(x1^2).to_polynomial()
x0*x2 + 2*x1*x2 + 11*x2^2

weight()
Return the weight of the given form if it is homogeneous (i.e. a modular form).

EXAMPLES:

sage: D = ModularForms(1,12).0; M = ModularFormsRing(1)
sage: M(D).weight()
12
sage: M.zero().weight()
0
sage: e4 = ModularForms(1,4).0
sage: (M(D)+e4).weight()
Traceback (most recent call last):
...
ValueError: the given graded form is not homogeneous (not a modular form)

weights_list()
Return the list of the weights of all the homogeneous components of the given graded modular form.

EXAMPLES:
sage: M = ModularFormsRing(1)
sage: f4 = ModularForms(1, 4).0; f6 = ModularForms(1, 6).0; f8 = ModularForms(1, 8).0
sage: F4 = M(f4); F6 = M(f6); F8 = M(f8)
sage: F = F4 + F6 + F8
sage: F.weights_list()
[4, 6, 8]
sage: M(0).weights_list()
[0]

class sage.modular.modform.element.ModularFormElement(parent, x, check=True)

Bases: ModularForm_abstract, HeckeModuleElement

An element of a space of modular forms.

INPUT:

• parent - ModularForms (an ambient space of modular forms)
• x - a vector on the basis for parent
• check - if check is True, check the types of the inputs.

OUTPUT:

• ModularFormElement - a modular form

EXAMPLES:

sage: M = ModularForms(Gamma0(11), 2)
sage: f = M.0
sage: f.parent()
Modular Forms space of dimension 2 for Congruence Subgroup Gamma0(11) of weight 2 over Rational Field

atkin_lehner_eigenvalue(d=None, embedding=None)

Return the result of the Atkin-Lehner operator \( W_d \) on \( \text{self} \).

INPUT:

• \( d \) - a positive integer exactly dividing the level \( N \) of \( \text{self} \), i.e. \( d \) divides \( N \) and is coprime to \( N/d \).
  (Default: \( d = N \))

• embedding - ignored (but accepted for compatibility with \( \text{Newform} \).

atkin_lehner_eigenvalue()\)

OUTPUT:

The Atkin-Lehner eigenvalue of \( W_d \) on \( \text{self} \). If \( \text{self} \) is not an eigenform for \( W_d \), a \( \text{ValueError} \) is raised.

See also:

For the conventions used to define the operator \( W_d \), see \( \text{sage.modular.hecke.module. HeckeModule_free_module.atkin_lehner_operator}() \).  

EXAMPLES:

sage: CuspForms(1, 30).0.atkin_lehner_eigenvalue()
1
sage: CuspForms(2, 8).0.atkin_lehner_eigenvalue()
twist \( \chi, \text{level}=\text{None} \)

Return the twist of the modular form \( \text{self} \) by the Dirichlet character \( \chi \).

If \( \text{self} \) is a modular form \( f \) with character \( \epsilon \) and \( q \)-expansion

\[
f(q) = \sum_{n=0}^{\infty} a_n q^n,
\]

then the twist by \( \chi \) is a modular form \( f_{\chi} \) with character \( \epsilon \chi^2 \) and \( q \)-expansion

\[
f_{\chi}(q) = \sum_{n=0}^{\infty} \chi(n) a_n q^n.
\]

INPUT:

- \( \chi \) – a Dirichlet character
- \( \text{level} \) – (optional) the level \( N \) of the twisted form. By default, the algorithm chooses some not necessarily minimal value for \( N \) using [AL1978], Proposition 3.1. (See also [Kob1993], Proposition III.3.17, for a simpler but slightly weaker bound.)

OUTPUT:

The form \( f_{\chi} \) as an element of the space of modular forms for \( \Gamma_1(N) \) with character \( \epsilon \chi^2 \).

EXAMPLES:

```
sage: f = CuspForms(11, 2).0
sage: f.parent()
Cuspidal subspace of dimension 1 of Modular Forms space of dimension 2 for Congruence Subgroup Gamma0(11) of weight 2 over Rational Field
sage: f.q_expansion(6)
q - 2*q^2 - q^3 + 2*q^4 + q^5 + O(q^6)
sage: eps = DirichletGroup(3).0
sage: eps.parent()
Group of Dirichlet characters modulo 3 with values in Cyclotomic Field of order 2 and degree 1
sage: f_eps = f.twist(eps)
sage: f_eps.parent()
Cuspidal subspace of dimension 9 of Modular Forms space of dimension 16 for Congruence Subgroup Gamma0(99) of weight 2 over Cyclotomic Field of order 2 and degree 1
sage: f_eps.q_expansion(6)
q + 2*q^2 + 2*q^4 - q^5 + O(q^6)
```

Modular forms without character are supported:

```
sage: M = ModularForms(Gamma1(5), 2)
sage: f = M.gen(0); f
1 + 60*q^3 - 120*q^4 + 240*q^5 + O(q^6)
```

(continues on next page)
The base field of the twisted form is extended if necessary:

```
sage: chi = DirichletGroup(2)[0]
sage: f.twist(chi)
60*q^3 + 240*q^5 + O(q^6)
```

REFERENCES:
- [AL1978]
- [Kob1993]

AUTHORS:
- L. J. P. Kilford (2009-08-28)
- Peter Bruin (2015-03-30)

```
class sage.modular.modform.element.ModularFormElement_elliptic_curve(parent, E)

Bases: Newform

A modular form attached to an elliptic curve over $\mathbb{Q}$.

\texttt{atkin\_lehner\_eigenvalue}(d=None, embedding=None)

Return the result of the Atkin-Lehner operator $W_d$ on $\text{self}$.

\textbf{INPUT:}
- $d$ – a positive integer exactly dividing the level $N$ of $\text{self}$, i.e. $d$ divides $N$ and is coprime to $N/d$. (Defaults to $d = N$ if not given.)
- $\text{embedding}$ – ignored (but accepted for compatibility with \texttt{Newform.atkin\_lehner\_action()})

\textbf{OUTPUT:}
The Atkin-Lehner eigenvalue of $W_d$ on $\text{self}$. This is either 1 or $-1$.

\textbf{EXAMPLES:}
```
sage: EllipticCurve('57a1').newform().atkin_lehner_eigenvalue()
1
sage: EllipticCurve('57b1').newform().atkin_lehner_eigenvalue()
-1
sage: EllipticCurve('57b1').newform().atkin_lehner_eigenvalue(19)
1
```
elliptic_curve()

Return elliptic curve associated to self.

EXAMPLES:

```
sage: E = EllipticCurve('11a')
sage: f = E.modular_form()
sage: f.elliptic_curve()
Elliptic Curve defined by y^2 + y = x^3 - x^2 - 10*x - 20 over Rational Field

sage: f.elliptic_curve() is E
True
```

class sage.modular.modform.element.ModularForm_abstract

Bases: ModuleElement

Constructor for generic class of a modular form. This should never be called directly; instead one should instantiate one of the derived classes of this class.

atkin_lehner_eigenvalue(d=None, embedding=None)

Return the eigenvalue of the Atkin-Lehner operator \( W_d \) acting on self.

INPUT:

- d – a positive integer exactly dividing the level \( N \) of self, i.e. \( d \) divides \( N \) and is coprime to \( N/d \) (default: \( d = N \))
- embedding – (optional) embedding of the base ring of self into another ring

OUTPUT:

The Atkin-Lehner eigenvalue of \( W_d \) on self. This is returned as an element of the codomain of embedding if specified, and in (a suitable extension of) the base field of self otherwise.

If self is not an eigenform for \( W_d \), a ValueError is raised.

See also:

sage.modular.hecke.module.HeckeModule_free_module.atkin_lehner_operator() (especially for the conventions used to define the operator \( W_d \)).

EXAMPLES:

```
sage: CuspForms(1, 12).0.atkin_lehner_eigenvalue()
1
sage: CuspForms(2, 8).0.atkin_lehner_eigenvalue()
Traceback (most recent call last):
...
NotImplementedError: don't know how to compute Atkin-Lehner matrix acting on \( \Lambda \) this space (try using a newform constructor instead)
```

character(compute=True)

Return the character of self. If compute=False, then this will return None unless the form was explicitly created as an element of a space of forms with character, skipping the (potentially expensive) computation of the matrices of the diamond operators.

EXAMPLES:

```
sage: ModularForms(DirichletGroup(17).0^2,2).2.character()
Dirichlet character modulo 17 of conductor 17 mapping 3 |---> zeta8
```

(continues on next page)
sage: CuspForms(Gamma1(7), 3).gen(0).character()  
Dirichlet character modulo 7 of conductor 7 mapping 3 |--> -1  
sage: CuspForms(Gamma1(7), 3).gen(0).character(compute = False) is None  
True  
sage: M = CuspForms(Gamma1(7), 5).gen(0).character()  
Traceback (most recent call last):  
  ...  
ValueError: Form is not an eigenvector for <3>

\textbf{cm\_discriminant}()  
\noindent Return the discriminant of the CM field associated to this form. An error will be raised if the form isn't of CM type.  
\textbf{EXAMPLES:}  
sage: Newforms(49, 2)[0].cm\_discriminant()  
-7  
sage: CuspForms(1, 12).gen(0).cm\_discriminant()  
Traceback (most recent call last):  
  ...  
ValueError: Not a CM form

\textbf{coefficient}(n)  
\noindent Return the n-th coefficient of the \(q\)-expansion of self.  
\textbf{INPUT:}  
\begin{itemize}  
\item n (int, Integer) - A non-negative integer.  
\end{itemize}  
\textbf{EXAMPLES:}  
sage: f = ModularForms(1, 12).0; f  
q - 24*q^2 + 252*q^3 - 1472*q^4 + 4830*q^5 + O(q^6)  
sage: f.coefficient(0)  
0  
sage: f.coefficient(1)  
1  
sage: f.coefficient(2)  
-24  
sage: f.coefficient(3)  
252  
sage: f.coefficient(4)  
-1472

\textbf{coefficients}(X)  
\noindent The coefficients \(a_n\) of self, for integers \(n\geq 0\) in the list \(X\). If \(X\) is an Integer, return coefficients for indices from 1 to \(X\).  
This function caches the results of the compute function.

\textbf{group}()  
\noindent Return the group for which \texttt{self} is a modular form.  
\textbf{EXAMPLES:}
```
sage: ModularForms(Gamma1(11), 2).gen(0).group()
Congruence Subgroup Gamma1(11)
```

**has_cm()**

Return whether the modular form `self` has complex multiplication.

**OUTPUT:**

Boolean

See also:

- `cm_discriminant()` (to return the CM field)
- `sage.schemes.elliptic_curves.ell_rational_field.has_cm()`

**EXAMPLES:**

```
sage: G = DirichletGroup(21); eps = G.0 * G.1
sage: Newforms(eps, 2)[0].has_cm()
True
```

This example illustrates what happens when `candidate_characters(self)` is the empty list.

```
sage: M = ModularForms(Gamma0(1), 12)
sage: C = M.cuspidal_submodule()
sage: Delta = C.gens()[0]
sage: Delta.has_cm()
False
```

We now compare the function `has_cm` between elliptic curves and their associated modular forms.

```
sage: E = EllipticCurve([-1, 0])
sage: f = E.modular_form()
sage: f.has_cm()
True
sage: E.has_cm() == f.has_cm()
True
```

Here is a non-cm example coming from elliptic curves.

```
sage: E = EllipticCurve('11a')
sage: f = E.modular_form()
sage: f.has_cm()
False
sage: E.has_cm() == f.has_cm()
True
```

**is_homogeneous()**

Return True.

For compatibility with elements of a graded modular forms ring.

An alias of this method is `is_modular_form`.

See also:

- `sage.modular.modform.element.GradedModularFormElement.is_homogeneous()`
EXAMPLES:

```python
sage: ModularForms(1,12).0.is_homogeneous()
True
```

### `is_modular_form()`
Return True.

For compatibility with elements of a graded modular forms ring.

An alias of this method is `is_modular_form`.

See also:
`sage.modular.modform.element.GradedModularFormElement.is_homogeneous()`

EXAMPLES:

```python
sage: ModularForms(1,12).0.is_homogeneous()
True
```

### `level()`
Return the level of self.

EXAMPLES:

```python
sage: ModularForms(25,4).0.level()
25
```

### `lseries(embedding=0, prec=53, max_imaginary_part=0, max_asymptotic_coeffs=40)`
Return the L-series of the weight k cusp form \( f \) on \( \Gamma_0(N) \).

This actually returns an interface to Tim Dokchitser’s program for computing with the L-series of the cusp form.

INPUT:

- `embedding` - either an embedding of the coefficient field of self into \( \mathbb{C} \), or an integer \( i \) between 0 and \( D-1 \) where \( D \) is the degree of the coefficient field (meaning to pick the \( i \)-th embedding). (Default: 0)
- `max_imaginary_part` - real number. Default: 0.
- `max_asymptotic_coeffs` - integer. Default: 40.

For more information on the significance of the last three arguments, see `dokchitser`.

Note: If an explicit embedding is given, but this embedding is specified to smaller precision than `prec`, it will be automatically refined to precision `prec`.

OUTPUT:

The L-series of the cusp form, as a `sage.lfunctions.dokchitser.Dokchitser` object.

EXAMPLES:

```python
sage: f = CuspForms(2,8).newforms()[0]
sage: L = f.lseries()
sage: L
```
L-series associated to the cusp form q – 8*q^2 + 12*q^3 + 64*q^4 – 210*q^5 + O(q^6)
sage: L(1)
0.0884317737041015
sage: L(0.5)
0.0296568512531983

As a consistency check, we verify that the functional equation holds:
sage: abs(L.check_functional_equation()) < 1.0e-20
True

For non-rational newforms we can specify an embedding of the coefficient field:
sage: f = Newforms(43, names='a')[1]
sage: K = f.hecke_eigenvalue_field()
sage: phi1, phi2 = K.embeddings(CC)
sage: L = f.lseries(embedding=phi1)
sage: L
L-series associated to the cusp form q + a1*q^2 - a1*q^3 + (-a1 + 2)*q^5 + O(q^6), a1=-1.41421356237310
sage: L(1)
0.620539857407845
sage: L = f.lseries(embedding=1)
sage: L(1)
0.921328017272472

An example with a non-real coefficient field (Q(ζ3) in this case):
sage: f = Newforms(Gamma1(13), 2, names='a')[0]
sage: f.lseries(embedding=0)(1)
0.298115272465799 - 0.0402203326076734*I
sage: f.lseries(embedding=1)(1)
0.298115272465799 + 0.0402203326076732*I

We compute with the L-series of the Eisenstein series $E_4$:
sage: f = ModularForms(1,4).0
sage: L = f.lseries()
sage: L(1)
-0.0304484570583933
sage: L = eisenstein_series_lseries(4)
sage: L(1)
-0.0304484570583933

Consistency check with delta_lseries (which computes coefficients in pari):
sage: delta = CuspForms(1,12).0
sage: L = delta.lseries()
sage: L(1)
0.0374412812685155
sage: L = delta_lseries()

(continues on next page)
We check that github issue #5262 is fixed:

```python
sage: E = EllipticCurve('37b2')
sage: h = Newforms(37)[1]
sage: LE = E.lseries()
```

```python
sage: Lh(1), LE(1)
(0.725681061936153, 0.725681061936153)
```

We check that github issue #25369 is fixed:

```python
sage: f5 = Newforms(Gamma1(4), 5, names='a')[0]; f5
q - 4*q^2 + 16*q^4 - 14*q^5 + O(q^6)
```

```python
sage: L5 = f5.lseries()
sage: abs(L5.check_functional_equation()) < 1e-15
True
```

We can change the precision (in bits):

```python
sage: f = Newforms(389, names='a')[0]
sage: L = f.lseries(prec=30)
sage: abs(L(1)) < 2^-30
True
```

```python
sage: L = f.lseries(prec=53)
sage: abs(L(1)) < 2^-53
True
```

```python
sage: L = f.lseries(prec=100)
sage: abs(L(1)) < 2^-100
True
```

```python
sage: f = Newforms(27, names='a')[0]
sage: L = f.lseries()
sage: L(1)
0.588879583428483
```

```
```

```python
sage: CuspForms(1,12).0.padded_list(20)
[0, 1, -24, 252, -1472, 4830, -6048, -16744, 84480, -113643,
-115920, 534612, -370944, -577738, 401856, 1217160, 987136,
-6905934, 2727432, 10661420]
```

```
```

```
```

```python
sage: period(M, prec=53)
```

Return the period of self with respect to M.
INPUT:

- self – a cusp form \( f \) of weight 2 for \( \Gamma_0(N) \)
- \( \mathcal{M} \) – an element of \( \Gamma_0(N) \)
- prec – (default: 53) the working precision in bits. If \( f \) is a normalised eigenform, then the output is correct to approximately this number of bits.

OUTPUT:

A numerical approximation of the period \( P_f(M) \). This period is defined by the following integral over the complex upper half-plane, for any \( \alpha \) in \( \mathbb{P}^1(\mathbb{Q}) \):

\[
P_f(M) = 2\pi i \int_{\alpha}^{M(\alpha)} f(z) \, dz.
\]

This is independent of the choice of \( \alpha \).

EXAMPLES:

```python
sage: C = Newforms(11, 2)[0]
sage: m = C.group()(matrix([[-4, -3], [11, 8]]))
sage: C.period(m)
-0.634604652139776 - 1.45881661693850*I
sage: f = Newforms(15, 2)[0]
sage: g = Gamma0(15)(matrix([[-4, -3], [15, 11]]))
sage: f.period(g)  # abs tol 1e-15
2.17298044293747e-16 - 1.59624222213178*I
```

If \( E \) is an elliptic curve over \( \mathbb{Q} \) and \( f \) is the newform associated to \( E \), then the periods of \( f \) are in the period lattice of \( E \) up to an integer multiple:

```python
sage: E = EllipticCurve('11a3')
sage: f = E.newform()
sage: g = Gamma0(11)([3, 1, 11, 4])
sage: f.period(g)
0.634604652139777 + 1.45881661693850*I
sage: omegal, omega2 = E.period_lattice().basis()
sage: -2/5*omegal + omega2
0.634604652139777 + 1.45881661693850*I
```

The integer multiple is 5 in this case, which is explained by the fact that there is a 5-isogeny between the elliptic curves \( J_0(5) \) and \( E \).

The elliptic curve \( E \) has a pair of modular symbols attached to it, which can be computed using the method `sage.schemes.elliptic_curves.ell_rational_field.EllipticCurve_rational_field.modular_symbol()`. These can be used to express the periods of \( f \) as exact linear combinations of the real and the imaginary period of \( E \):

```python
sage: s = E.modular_symbol(sign=+1)
sage: t = E.modular_symbol(sign=-1, implementation="sage")
sage: s(3/11), t(3/11)
(1/10, 1/2)
sage: s(3/11)*omegal + t(3/11)*2*omega2.imag()*I
0.634604652139777 + 1.45881661693850*I
```
ALGORITHM:

We use the series expression from [Cre1997], Chapter II, Proposition 2.10.3. The algorithm sums the first $T$ terms of this series, where $T$ is chosen in such a way that the result would approximate $P_f(M)$ with an absolute error of at most $2^{-\text{prec}}$ if all computations were done exactly.

Since the actual precision is finite, the output is currently not guaranteed to be correct to $\text{prec}$ bits of precision.

\texttt{petersson\_norm(embedding=0, prec=53)}

Compute the Petersson scalar product of $f$ with itself:

$$\langle f, f \rangle = \int_{\Gamma_0(N) \backslash \mathbb{H}} |f(x + iy)|^2 y^k \, dx \, dy.$$  

Only implemented for $N = 1$ at present. It is assumed that $f$ has real coefficients. The norm is computed as a special value of the symmetric square L-function, using the identity

$$\langle f, f \rangle = \frac{(k-1)!L(\text{Sym}^2 f, k)}{2^{2k-1} \pi^{k+1}}$$

INPUT:

- \text{embedding}: embedding of the coefficient field into $\mathbb{R}$ or $\mathbb{C}$, or an integer $i$ (interpreted as the $i$-th embedding) (default: 0)
- \text{prec} (integer, default 53): precision in bits

EXAMPLES:

\begin{verbatim}
sage: CuspForms(1, 16).0.petersson_norm()
verbose -1 (...) Warning: Loss of 2 decimal digits due to cancellation
2.16906134759063e-6

The Petersson norm depends on a choice of embedding:

sage: set_verbose(-2, "dokchitser.py") # disable precision-loss warnings
sage: F = Newforms(1, 24, names='a')[0]
sage: F.petersson_norm(embedding=0)
0.000107836545077234
sage: F.petersson_norm(embedding=1)
0.000128992800758160
\end{verbatim}

\texttt{prec()}

Return the precision to which \text{self.q\_expansion()} is currently known. Note that this may be 0.

EXAMPLES:

\begin{verbatim}
sage: M = ModularForms(2,14)
sage: f = M.0
sage: f.prec()
0

sage: M.prec(20)
20
sage: f.prec()
0
sage: x = f.q_expansion() ; f.prec()
0
\end{verbatim}
The \( q \)-expansion of the modular form to precision \( O(q^{\text{prec}}) \). This function takes one argument, which is the integer \( \text{prec} \).

**EXAMPLES:**

We compute the cusp form \( \Delta \):

```
sage: delta = CuspForms(1,12).0
sage: delta.q_expansion()
q - 24*q^2 + 252*q^3 - 1472*q^4 + 4830*q^5 + O(q^6)
```

We compute the \( q \)-expansion of one of the cusp forms of level 23:

```
sage: f = CuspForms(23,2).0
sage: f.q_expansion()
q - q^3 - q^4 + O(q^6)
sage: f.q_expansion(10)
q - q^3 - q^4 - 2*q^6 + 2*q^7 - q^8 + 2*q^9 + O(q^10)
sage: f.q_expansion(2)
q + O(q^2)
sage: f.q_expansion(0)
O(q^0)
sage: f.q_expansion(-1)
Traceback (most recent call last):
  ...
ValueError: prec (= -1) must be non-negative
```

Same as \( \text{self}.q\_\text{expansion}(\text{prec}) \).

**See also:**

\( q\_\text{expansion}() \)

**EXAMPLES:**

```
sage: CuspForms(1,12).0.qexp()
q - 24*q^2 + 252*q^3 - 1472*q^4 + 4830*q^5 + O(q^6)
```

Return the Serre derivative of the given modular form.

If \( \text{self} \) is of weight \( k \), then the returned modular form will be of weight \( k + 2 \).

**EXAMPLES:**

```
sage: E4 = ModularForms(1, 4).0
sage: E6 = ModularForms(1, 6).0
sage: DE4 = E4.serre_derivative(); DE4
-1/3 + 168*q + 5544*q^2 + 40992*q^3 + 177576*q^4 + 525168*q^5 + O(q^6)
sage: DE6 = E6.serre_derivative(); DE6
-1/2 - 240*q - 30960*q^2 - 525120*q^3 - 3963120*q^4 - 18750240*q^5 + O(q^6)
sage: Del = ModularForms(1, 12).0 # Modular discriminant
sage: Del.serre_derivative()
```

(continues on next page)
The Serre derivative raises the weight of a modular form by $2$:

```python
sage: DE4.weight()
sage: DE6.weight()
sage: Df.weight()
```

The Ramanujan identities are verified (see Wikipedia article Eisenstein_series#Ramanujan_identities):

```python
sage: DE4 == (-1/3) * E6
sage: DE6 == (-1/2) * E4 * E4
```

**symsquare_lseries**(*chi=None, embedding=0, prec=53*)

Compute the symmetric square L-series of this modular form, twisted by the character $\chi$.

**INPUT:**

- `chi` – Dirichlet character to twist by, or None (default None, interpreted as the trivial character).
- `embedding` – embedding of the coefficient field into $\mathbb{R}$ or $\mathbb{C}$, or an integer $i$ (in which case take the $i$-th embedding)
- `prec` – The desired precision in bits (default 53).

**OUTPUT:** The symmetric square L-series of the cusp form, as a `sage.lfunctions.dokchitser.Dokchitser` object.

**EXAMPLES:**

```python
sage: CuspForms(1, 12).0.symsquare_lseries()(22)
sage: psi = DirichletGroup(7).0^2
sage: L = CuspForms(1, 16).0.symsquare_lseries(psi)
```

An example with coefficients not in $\mathbb{Q}$:

```python
sage: F = Newforms(1, 24, names='a')[0]
sage: K = F.hecke_eigenvalue_field()
sage: phi = K.embeddings(RR)[0]
sage: L = F.symsquare_lseries(embedding=phi)
```
AUTHORS:
- Martin Raum (2011) – original code posted to sage-nt
- David Loeffler (2015) – added support for twists, integrated into Sage library

valuation()
Return the valuation of self (i.e. as an element of the power series ring in q).

EXAMPLES:

```
sage: ModularForms(11,2).0.valuation()
sage: 1
sage: ModularForms(11,2).1.valuation()
sage: 0
sage: ModularForms(25,6).1.valuation()
sage: 2
sage: ModularForms(25,6).6.valuation()
sage: 7
```

weight()
Return the weight of self.

EXAMPLES:

```
sage: (ModularForms(Gamma1(9),2).6).weight()
sage: 2
```

```
class sage.modular.modform.element.Newform(parent, component, names, check=True)

Initialize a Newform object.

INPUT:
- parent - An ambient cuspidal space of modular forms for which self is a newform.
- component - A simple component of a cuspidal modular symbols space of any sign corresponding to this newform.
- check - If check is True, check that parent and component have the same weight, level, and character, that component has sign 1 and is simple, and that the types are correct on all inputs.

EXAMPLES:

```
sage: sage.modular.modform.element.Newform(CuspForms(11,2), ModularSymbols(11,2, sign=1).cuspidal_subspace(), 'a')
q - 2*q^2 - q^3 + 2*q^4 + q^5 + O(q^6)

sage: f = Newforms(DirichletGroup(5).0, 7,names='a')[0]; f[2].trace(f.base_ring().base_field())
-5*zeta4 - 5
```
abelian_variety()

Return the abelian variety associated to self.

EXAMPLES:

```
sage: Newforms(14,2)[0]
q - q^2 - 2*q^3 + q^4 + O(q^6)
sage: Newforms(14,2)[0].abelian_variety()
Newform abelian subvariety 14a of dimension 1 of J0(14)
sage: Newforms(1, 12)[0].abelian_variety()
Traceback (most recent call last):
... TypeError: f must have weight 2
```

atkin_lehner_action(d=None, normalization='analytic', embedding=None)

Return the result of the Atkin-Lehner operator $W_d$ on this form $f$, in the form of a constant $\lambda_d(f)$ and a normalized newform $f'$ such that

$$f \mid W_d = \lambda_d(f)f'.$$

See atkin_lehner_eigenvalue() for further details.

EXAMPLES:

```
sage: f = Newforms(DirichletGroup(30).1^2, 2, names='a')[0]
sage: emb = f.base_ring().complex_embeddings()[0]
sage: for d in divisors(30):
...:     print(f.atkin_lehner_action(d, embedding=emb))
(1.00000000000000, q + a0*q^2 - a0*q^3 - q^4 + (a0 - 2)*q^5 + O(q^6))
(-1.00000000000000*I, q + a0*q^2 - a0*q^3 - q^4 + (a0 - 2)*q^5 + O(q^6))
(-0.894427190999916 + 0.447213595499958*I, q - a0*q^2 + a0*q^3 - q^4 + (-a0 - →2)*q^5 + O(q^6))
(1.00000000000000, q + a0*q^2 - a0*q^3 - q^4 + (a0 - 2)*q^5 + O(q^6))
(-0.447213595499958 - 0.894427190999916*I, q - a0*q^2 + a0*q^3 - q^4 + (-a0 - →2)*q^5 + O(q^6))
(0.447213595499958 + 0.894427190999916*I, q - a0*q^2 + a0*q^3 - q^4 + (-a0 - →2)*q^5 + O(q^6))
(-0.447213595499958 + 0.447213595499958*I, q - a0*q^2 + a0*q^3 - q^4 + (-a0 - →2)*q^5 + O(q^6))
```

The above computation can also be done exactly:

```
sage: K.<z> = CyclotomicField(20)
sage: f = Newforms(DirichletGroup(30).1^2, 2, names='a')[0]
sage: emb = f.base_ring().embeddings(CyclotomicField(20, 'z'))[0]
sage: for d in divisors(30):
...:     print(f.atkin_lehner_action(d, embedding=emb))
(1, q + a0*q^2 - a0*q^3 - q^4 + (a0 - 2)*q^5 + O(q^6))
(z^5, q + a0*q^2 - a0*q^3 - q^4 + (a0 - 2)*q^5 + O(q^6))
(z^5, q + a0*q^2 - a0*q^3 - q^4 + (a0 - 2)*q^5 + O(q^6))
(-2/5*z^7 + 4/5*z^6 + 1/5*z^5 - 4/5*z^4 - 2/5*z^3 - 2/5, q - a0*q^2 + a0*q^3 - →q^4 + (-a0 - 2)*q^5 + O(q^6))
(1, q + a0*q^2 - a0*q^3 - q^4 + (a0 - 2)*q^5 + O(q^6))
(4/5*z^7 + 2/5*z^6 - 2/5*z^5 - 2/5*z^4 + 4/5*z^3 - 1/5, q - a0*q^2 + a0*q^3 - q^4 + →q^5 + O(q^6))
```

(continues on next page)
We can compute the eigenvalue of $W_p$ in certain cases where the $p$-th coefficient of $f$ is zero:

```python
sage: f = Newforms(169, names='a')[0]; f
q + a0*q^2 + 2*q^3 + q^4 - a0*q^5 + O(q^6)
sage: f[13]
0
sage: f.atkin_lehner_eigenvalue(169)
-1
```

An example showing the non-multiplicativity of the pseudo-eigenvalues:

```python
sage: chi = DirichletGroup(18).0^4
sage: f = Newforms(chi, 2)[0]

sage: w2, _ = f.atkin_lehner_action(2); w2
zeta6
sage: w9, _ = f.atkin_lehner_action(9); w9
-zeta18^4
sage: w18, _ = f.atkin_lehner_action(18); w18
-zeta18
sage: w18 == w2 * w9 * chi(crt(2, 9, 9, 2))
True
```

### atkin_lehner_eigenvalue

#### (d=None, normalization='analytic', embedding=None)

Return the pseudo-eigenvalue of the Atkin-Lehner operator $W_d$ acting on this form $f$.

**INPUT:**

- $d$ – a positive integer exactly dividing the level $N$ of $f$, i.e. $d$ divides $N$ and is coprime to $N/d$. The default is $d = N$.
  - If $d$ does not divide $N$ exactly, then it will be replaced with a multiple $D$ of $d$ such that $D$ exactly divides $N$ and $D$ has the same prime factors as $d$. An error will be raised if $d$ does not divide $N$.
- `normalization` – either 'analytic' (the default) or 'arithmetic'; see below.
- `embedding` – (optional) embedding of the coefficient field of $f$ into another ring. Ignored if `normalization = 'arithmetic'`.

**OUTPUT:**

The Atkin-Lehner pseudo-eigenvalue of $W_d$ on $f$, as an element of the coefficient field of $f$, or the codomain of embedding if specified.

As defined in [AL1978], the pseudo-eigenvalue is the constant $\lambda_d(f)$ such that

```
\mathbf{\lambda} d(f) f'
```

where $f'$ is some normalised newform (not necessarily equal to $f$).
If \texttt{normalisation='analytic'} (the default), this routine will compute $\lambda_d$, using the conventions of [AL1978] for the weight $k$ action, which imply that $\lambda_d$ has complex absolute value 1. However, with these conventions $\lambda_d$ is not in the Hecke eigenvalue field of $f$ in general, so it is often necessary to specify an embedding of the eigenvalue field into a larger ring (which needs to contain roots of unity of sufficiently large order, and a square root of $d$ if $k$ is odd).

If \texttt{normalisation='arithmetic'} we compute instead the quotient

.. math::
   \frac{d^{k/2-1} \lambda_d(f) \varepsilon_{N/d}(d / d_0)}{G(\varepsilon_d)},

where $G(\varepsilon_d)$ is the Gauss sum of the $d$-primary part of the nebentype of $f$ (more precisely, of its associated primitive character), and $d_0$ its conductor. This ratio is always in the Hecke eigenvalue field of $f$ (and can be computed using only arithmetic in this field), so specifying an embedding is not needed, although we still allow it for consistency.

(Note that if $k = 2$ and $\varepsilon$ is trivial, both normalisations coincide.)

\textbf{See also:}

- \texttt{sage.modular.hecke.module.atkin_lehner_operator()} (especially for the conventions used to define the operator $W_d$)
- \texttt{atkin_lehner_action()}, which returns both the pseudo-eigenvalue and the newform $f'$.

\textbf{EXAMPLES:}

```python
sage: [x.atkin_lehner_eigenvalue() for x in ModularForms(53).newforms('a')]
[1, -1]
```

```python
sage: f = Newforms(Gamma1(15), 3, names='a')[2]; f
q + a2*q^2 + (-a2 - 2)*q^3 - q^4 - a2*q^5 + O(q^6)
```

```python
sage: f.atkin_lehner_eigenvalue(5)
Traceback (most recent call last):
  ... ValueError: Unable to compute square root. Try specifying an embedding into a
    larger ring
```

```python
sage: L = f.hecke_eigenvalue_field(); x = polygen(QQ); M.<sqrt5> = L.extension(x^2 - 5)
```

```python
sage: f.atkin_lehner_eigenvalue(5, embedding=M.coerce_map_from(L))
1/5*a2*sqrt5
```

```python
sage: f.atkin_lehner_eigenvalue(5, normalization='arithmetic')
a2
```

\textbf{character()}

The nebentypus character of this newform (as a Dirichlet character with values in the field of Hecke eigenvalues of the form).

\textbf{EXAMPLES:}
coefficient\( (n) \)

Return the coefficient of \( q^n \) in the power series of self.

**INPUT:**

- \( n \) - a positive integer

**OUTPUT:**

- the coefficient of \( q^n \) in the power series of self.

**EXAMPLES:**

```
sage: f = Newforms(11)[0]; f
q - 2*q^2 - q^3 + 4*q^4 + q^5 + O(q^6)
sage: f.coefficient(100)
-8
```

```
sage: g = Newforms(23, names='a')[0]; g
q + a0*q^2 + (-2*a0 - 1)*q^3 + (-a0 - 1)*q^4 + 2*a0*q^5 + O(q^6)
sage: g.coefficient(3)
-2*a0 - 1
```

\[ \text{element}() \]

Find an element of the ambient space of modular forms which represents this newform.

**Note:** This can be quite expensive. Also, the polynomial defining the field of Hecke eigenvalues should be considered random, since it is generated by a random sum of Hecke operators. (The field itself is not random, of course.)

**EXAMPLES:**

```
sage: ls = Newforms(38,4,names='a')
sage: ls[0]
q - 2*q^2 - 2*q^3 + 4*q^4 - 9*q^5 + O(q^6)
sage: ls # random
[ q - 2*q^2 - 2*q^3 + 4*q^4 - 9*q^5 + O(q^6),
  q - 2*q^2 + (-a1 - 2)*q^3 + 4*q^4 + (2*a1 + 10)*q^5 + O(q^6),
  q + 2*q^2 + (1/2*a2 - 1)*q^3 + 4*q^4 + (-3/2*a2 + 12)*q^5 + O(q^6)]
sage: type(ls[0])
<class 'sage.modular.modform.element.Newform'>
sage: ls2 = [ x.element() for x in ls ]
sage: ls2 # random
[ x^2 - 9*x + 2  for x in ls ]
sage: ls2
[ q - 2*q^2 - 2*q^3 + 4*q^4 - 9*q^5 + O(q^6),
  q - 2*q^2 + (-a1 - 2)*q^3 + 4*q^4 + (2*a1 + 10)*q^5 + O(q^6),
  q + 2*q^2 + (1/2*a2 - 1)*q^3 + 4*q^4 + (-3/2*a2 + 12)*q^5 + O(q^6)]
sage: type(ls2[0])
```

(continues on next page)
hecke_eigenvalue_field()
Return the field generated over the rationals by the coefficients of this newform.

EXAMPLES:

```python
sage: ls = Newforms(35, 2, names='a') ; ls
[q + q^3 - 2*q^4 - q^5 + O(q^6),
q + a1*q^2 + (-a1 - 1)*q^3 + (-a1 + 2)*q^4 + q^5 + O(q^6)]
sage: ls[0].hecke_eigenvalue_field()
Rational Field
sage: ls[1].hecke_eigenvalue_field()
Number Field in a1 with defining polynomial x^2 + x - 4
```

is_cuspidal()
Return True. For compatibility with elements of modular forms spaces.

EXAMPLES:

```python
sage: Newforms(11, 2)[0].is_cuspidal()
True
```

local_component(p, twist_factor=None)
Calculate the local component at the prime $p$ of the automorphic representation attached to this newform. For more information, see the documentation of the LocalComponent() function.

EXAMPLES:

```python
sage: f = Newform("49a")
sage: f.local_component(7)
Smooth representation of GL_2(Q_7) with conductor 7^2
```

minimal_twist(p=None)
Compute a pair $(g, \chi)$ such that $g = f \otimes \chi$, where $f$ is this newform and $\chi$ is a Dirichlet character, such that $g$ has level as small as possible. If the optional argument $p$ is given, consider only twists by Dirichlet characters of $p$-power conductor.

EXAMPLES:

```python
sage: f = Newforms(121, 2)[3] # long time
sage: g, chi = f.minimal_twist(5) # long time
sage: g
q - 2*q^2 - q^3 + 2*q^4 + q^5 + O(q^6)
sage: chi
Dirichlet character modulo 11 of conductor 11 mapping 2 |--> -1
dl: f.twist(chi, level=11) == g
True
```
Modular Forms, Release 10.0

sage: g # long time
q + a*q^2 - a*q^3 - 2*q^4 + (1/2*a + 2)*q^5 + O(q^6)
sage: chi # long time
Dirichlet character modulo 5 of conductor 5 mapping 2 |--> 1/2*a
sage: f.twist(chi, level=g.level()) == g # long time
True

modsym_eigenspace(sign=0)

Return a submodule of dimension 1 or 2 of the ambient space of the sign 0 modular symbols space associated to self, base-extended to the Hecke eigenvalue field, which is an eigenspace for the Hecke operators with the same eigenvalues as this newform, and is an eigenspace for the star involution of the appropriate sign if the sign is not 0.

EXAMPLES:

sage: N = Newform("37a")
sage: N.modular_symbols(0)
Modular Symbols subspace of dimension 2 of Modular Symbols space of dimension 5 → for Gamma_0(37) of weight 2 with sign 0 over Rational Field
sage: V = N.modsym_eigenspace(1); V
Vector space of degree 5 and dimension 1 over Rational Field
Basis matrix:
[ 0 1 -1 1 0]
sage: V.0 in M.free_module()
True
sage: V = N.modsym_eigenspace(-1); V
Vector space of degree 5 and dimension 1 over Rational Field
Basis matrix:
[ 0 0 0 1 -1/2]
sage: V.0 in M.free_module()
True

modular_symbols(sign=0)

Return the subspace with the specified sign of the space of modular symbols corresponding to this newform.

EXAMPLES:

sage: f = Newforms(18,4)[0]
sage: f.modular_symbols() Modular Symbols subspace of dimension 2 of Modular Symbols space of dimension 18 for Gamma_0(18) of weight 4 with sign 0 over Rational Field
sage: f.modular_symbols(1)
Modular Symbols subspace of dimension 1 of Modular Symbols space of dimension 18 for Gamma_0(18) of weight 4 with sign 1 over Rational Field

number()

Return the index of this space in the list of simple, new, cuspidal subspaces of the full space of modular symbols for this weight and level.

EXAMPLES:

sage: Newforms(43, 2, names='a')[1].number()
1

1.13. Elements of modular forms spaces
twist(chi, level=None, check=True)

Return the twist of the newform self by the Dirichlet character chi.

If self is a newform \( f \) with character \( \epsilon \) and \( q \)-expansion

\[
f(q) = \sum_{n=1}^{\infty} a_n q^n,
\]

then the twist by \( \chi \) is the unique newform \( f \otimes \chi \) with character \( \epsilon \chi^2 \) and \( q \)-expansion

\[
(f \otimes \chi)(q) = \sum_{n=1}^{\infty} b_n q^n
\]

satisfying \( b_n = \chi(n) a_n \) for all but finitely many \( n \).

INPUT:

- \( \chi \) – a Dirichlet character. Note that Sage must be able to determine a common base field into which both the Hecke eigenvalue field of self, and the field of values of \( \chi \), can be embedded.

- \( \text{level} \) – (optional) the level \( N \) of the twisted form. If \( N \) is not given, the algorithm tries to compute \( N \) using [AL1978], Theorem 3.1; if this is not possible, it returns an error. If \( N \) is given but incorrect, i.e. the twisted form does not have level \( N \), then this function will attempt to detect this and return an error, but it may sometimes return an incorrect answer (a newform of level \( N \) whose first few coefficients agree with those of \( f \otimes \chi \)).

- \( \text{check} \) – (optional) boolean; if True (default), ensure that the space of modular symbols that is computed is genuinely simple and new. This makes it less likely, but not impossible, that a wrong result is returned if an incorrect \( \text{level} \) is specified.

OUTPUT:

The form \( f \otimes \chi \) as an element of the set of newforms for \( \Gamma_1(N) \) with character \( \epsilon \chi^2 \).

EXAMPLES:

```sage
G = DirichletGroup(3, base_ring=QQ)
sage: Delta = Newforms(SL2Z, 12)[0]; Delta
q - 24*q^2 + 252*q^3 - 1472*q^4 + 4830*q^5 + O(q^6)
sage: Delta.twist(G[0]) == Delta
True
sage: Delta.twist(G[1])  # long time (about 5 s)
q + 24*q^2 - 1472*q^4 - 4830*q^5 + O(q^6)

sage: M = CuspForms(Gamma1(13), 2)
sage: f = M.newforms('a')[0]; f
q + a0*q^2 + (-2*a0 - 4)*q^3 + (-a0 - 1)*q^4 + (2*a0 + 3)*q^5 + O(q^6)
sage: f.twist(G[1])
q - a0*q^2 + (-a0 - 1)*q^4 + (-2*a0 - 3)*q^5 + O(q^6)

sage: f = Newforms(Gamma1(30), 2, names='a')[1]; f
q + a1*q^2 - a1*q^3 - q^4 + (a1 - 2)*q^5 + O(q^6)
sage: f.twist(f.character())
Traceback (most recent call last):
... Not Implemented Error: cannot calculate 5-primary part of the level of the twist of q + a1*q^2 - a1*q^3 - q^4 + (a1 - 2)*q^5 + O(q^6) by Dirichlet character
```

(continues on next page)
→ modulo 5 of conductor 5 mapping 2 |--> -1

```sage
f = ModularForm(5, 2, 5)
sage: f.twist(f.character(), level=30)
q - a1*q^2 + a1*q^3 - q^4 + (-a1 - 2)*q^5 + O(q^6)
```

AUTHORS:

- Peter Bruin (April 2015)

```sage
sage.modular.modform.element.delta_lseries(prec=53, max_imaginary_part=0, max_asympt_coeffs=40, algorithm=None)
```

Return the L-series of the modular form $\Delta$.

If algorithm is “gp”, this returns an interface to Tim Dokchitser’s program for computing with the L-series of the modular form $\Delta$.

If algorithm is “pari”, this returns instead an interface to Pari’s own general implementation of L-functions.

**INPUT:**

- `prec` – integer (bits precision)
- `max_imaginary_part` – real number
- `max_asympt_coeffs` – integer
- `algorithm` – optional string: 'gp' (default), 'pari'

**OUTPUT:**

The L-series of $\Delta$.

**EXAMPLES:**

```sage
sage: L = delta_lseries()
sage: L(1)
0.0374412812685155
sage: L = delta_lseries(algorithm='pari')
sage: L(1)
0.0374412812685155
```

```sage
sage.modular.modform.element.is_ModularFormElement(x)
```

Return True if x is a modular form.

**EXAMPLES:**

```sage
sage: from sage.modular.modform.element import is_ModularFormElement
sage: is_ModularFormElement(5)
False
sage: is_ModularFormElement(ModularForms(11).0)
True
```
1.14 Hecke operators on \( q \)-expansions

\[
\text{sage.modular.modform.hecke_operator_on_qexp.hecke_operator_on_basis}(B, n, k, eps=None, already_echelonized=False)
\]

Given a basis \( B \) of \( q \)-expansions for a space of modular forms with character \( \varepsilon \) to precision at least \( \#B \cdot n + 1 \), this function computes the matrix of \( T_n \) relative to \( B \).

**Note:** If the elements of \( B \) are not known to sufficient precision, this function will report that the vectors are linearly dependent (since they are to the specified precision).

**INPUT:**
- \( B \) - list of \( q \)-expansions
- \( n \) - an integer \( \geq 1 \)
- \( k \) - an integer
- \( \varepsilon \) - Dirichlet character
- \( \text{already_echelonized} \) – bool (default: False); if True, use that the basis is already in Echelon form, which saves a lot of time.

**EXAMPLES:**

```python
sage: sage.modular.modform.constructor.ModularForms_clear_cache()

sage: ModularForms(1,12).q_expansion_basis()  # Output is truncated for clarity
\[
q - 24q^2 + 252q^3 - 1472q^4 + 4830q^5 + O(q^6),
1 + 65520/691q + 134250480/691q^2 + 11606736960/691q^3 + 274945048560/691q^4 +
\cdots
3199218815520/691q^5 + O(q^6)
\]

sage: hecke_operator_on_basis(ModularForms(1,12).q_expansion_basis(), 3, 12)
Traceback (most recent call last):
... 
ValueError: The given basis vectors must be linearly independent.

sage: hecke_operator_on_basis(ModularForms(1,12).q_expansion_basis(30), 3, 12)
\[
[ 252 0]
[ 0 177148]
\]
```

\[
\text{sage.modular.modform.hecke_operator_on_qexp.hecke_operator_on_qexp}(f, n, k, eps=None, prec=None, check=True, _return_list=False)
\]

Given the \( q \)-expansion \( f \) of a modular form with character \( \varepsilon \), this function computes the image of \( f \) under the Hecke operator \( T_{n,k} \) of weight \( k \).

**EXAMPLES:**

```python
sage: M = ModularForms(1,12)

sage: hecke_operator_on_qexp(M.basis()[0], 3, 12)
252*q - 6048*q^2 + 63504*q^3 - 370944*q^4 + O(q^5)

sage: hecke_operator_on_qexp(M.basis()[0], 1, 12, prec=7)
q - 24*q^2 + 252*q^3 - 1472*q^4 + 4830*q^5 - 6048*q^6 + O(q^7)

sage: hecke_operator_on_qexp(M.basis()[0], 1, 12)
```

(continues on next page)
q - 24*q^2 + 252*q^3 - 1472*q^4 + 4830*q^5 - 6048*q^6 - 16744*q^7 + 84480*q^8 -
˓→113643*q^9 - 115920*q^10 + 534612*q^11 - 370944*q^12 - 577738*q^13 + O(q^14)

sage: M.prec(20)
20

sage: hecke_operator_on_qexp(M.basis()[0], 3, 12)
252*q - 6048*q^2 + 63504*q^3 - 6048*q^4 + 1217160*q^5 - 1524096*q^6 + O(q^7)

sage: hecke_operator_on_qexp(M.basis()[0], 1, 12)
q - 24*q^2 + 252*q^3 - 6048*q^4 + 4830*q^5 - 1472*q^6 + 113643*q^7 - 115920*q^8 -
˓→534612*q^9 + 577738*q^10 + 84480*q^11 - 6048*q^12 + 63504*q^13 + 401856*q^14 -
˓→7109760*q^15 + 987136*q^16 - 6905934*q^17 + 2727432*q^18 + 10661420*q^19 -
˓→7109760*q^20 + O(q^21)

sage: (hecke_operator_on_qexp(M.basis()[0], 1, 12)*252).add_bigoh(7)
252*q - 6048*q^2 + 63504*q^3 - 370944*q^4 + 1217160*q^5 - 1524096*q^6 + O(q^7)

sage: hecke_operator_on_qexp(M.basis()[0], 6, 12)
-6048*q + 145152*q^2 - 1524096*q^3 + O(q^4)

An example on a formal power series:

sage: R.<q> = QQ[[[]]

sage: f = q + q^2 + q^3 + q^7 + O(q^8)

sage: hecke_operator_on_qexp(f, 3, 12)
q + O(q^3)

sage: hecke_operator_on_qexp(delta_qexp(24), 3, 12).prec()
8

sage: hecke_operator_on_qexp(delta_qexp(25), 3, 12).prec()
9

An example of computing \( T_{p,k} \) in characteristic \( p \):

sage: p = 199

sage: fp = delta_qexp(prec=p^2+1, K=GF(p))

sage: tfp = hecke_operator_on_qexp(fp, p, 12)

sage: tfp == fp[p] * fp
True

sage: tf = hecke_operator_on_qexp(delta_qexp(prec=p^2+1), p, 12).change_ring(GF(p))

sage: tfp == tf
True

1.15 Numerical computation of newforms

class sage.modular.modform.numerical.NumericalEigenforms(group, weight=2, eps=1e-20, delta=0.01, tp=[2, 3, 5])

Bases: SageObject

numerical_eigenforms(group, weight=2, eps=1e-20, delta=1e-2, tp=[2,3,5])

INPUT:

- group - a congruence subgroup of a Dirichlet character of order 1 or 2
• \textbf{weight} - an integer $\geq 2$

• \textbf{eps} - a small float; $\text{abs(\ )} < \text{eps}$ is what “equal to zero” is interpreted as for floating point numbers.

• \textbf{delta} - a small-ish float; eigenvalues are considered distinct if their difference has absolute value at least delta

• \textbf{tp} - use the Hecke operators $T_p$ for $p$ in \texttt{tp} when searching for a random Hecke operator with distinct Hecke eigenvalues.

\textbf{OUTPUT:}

A numerical eigenforms object, with the following useful methods:

• \textbf{ap()} - return all eigenvalues of $T_p$

• \textbf{eigenvalues()} - list of eigenvalues corresponding to the given list of primes, e.g.:

\begin{verbatim}
[[eigenvalues of T_2],
 [eigenvalues of T_3],
 [eigenvalues of T_5], ...]
\end{verbatim}

• \textbf{systems_of_eigenvalues()} - a list of the systems of eigenvalues of eigenforms such that the chosen random linear combination of Hecke operators has multiplicity 1 eigenvalues.

\textbf{EXAMPLES:}

\begin{verbatim}
sage: n = numerical_eigenforms(23)
sage: n == loads(dumps(n))
True
sage: n.ap(2)  # abs tol 1e-12
[3.0, -1.6180339887498947, 0.6180339887498968]
sage: n.systems_of_eigenvalues(7)  # abs tol 2e-12
[
 [-1.6180339887498947, 2.2360679774997894, -3.2360679774997894],
 [0.6180339887498968, -2.2360679774997888, 1.2360679774997936],
 [3.0, 4.0, 6.0]]
sage: n.systems_of_abs(7)  # abs tol 2e-12
[[0.6180339887498968, 2.2360679774997888, 1.2360679774997936],
 [1.6180339887498947, 2.2360679774997894, 3.2360679774997894],
 [3.0, 4.0, 6.0]]
sage: n.eigenvalues([2,3,5])  # rel tol 2e-12
[[3.0, -1.6180339887498947, 0.6180339887498968],
 [4.0, 2.2360679774997894, -2.2360679774997888],
 [6.0, -3.2360679774997894, 1.2360679774997936]]
\end{verbatim}

\textbf{ap(p)}

Return a list of the eigenvalues of the Hecke operator $T_p$ on all the computed eigenforms. The eigenvalues match up between one prime and the next.

\textbf{INPUT:}

• \texttt{p} - integer, a prime number

\textbf{OUTPUT:}

• \texttt{list} - a list of double precision complex numbers
EXAMPLES:

```python
sage: n = numerical_eigenforms(11, 4)
sage: n.ap(2) # random order
[9.0, 9.0, 2.73205080757, -0.732050807569]
sage: n.ap(3) # random order
[28.0, 28.0, -7.92820323028, 5.92820323028]
sage: m = n.modular_symbols()
sage: x = polygen(QQ, 'x')
sage: m.T(2).charpoly(x).factor()
(x - 9)^2 * (x^2 - 2*x - 2)
sage: m.T(3).charpoly(x).factor()
(x - 28)^2 * (x^2 + 2*x - 47)
```

eigenvalues(primes)

Return the eigenvalues of the Hecke operators corresponding to the primes in the input list of primes. The eigenvalues match up between one prime and the next.

INPUT:

- primes - a list of primes

OUTPUT:

list of lists of eigenvalues.

EXAMPLES:

```python
sage: n = numerical_eigenforms(1, 12)
sage: n.eigenvalues([3, 5, 13])
```

level()

Return the level of this set of modular eigenforms.

EXAMPLES:

```python
sage: n = numerical_eigenforms(61) ; n.level()
61
```

modular_symbols()

Return the space of modular symbols used for computing this set of modular eigenforms.

EXAMPLES:

```python
sage: n = numerical_eigenforms(61) ; n.modular_symbols()
Modular Symbols space of dimension 5 for Gamma_0(61) of weight 2 with sign 1 over Rational Field
```

systems_of_abs(bound)

Return the absolute values of all systems of eigenvalues for self for primes up to bound.

EXAMPLES:

```python
sage: numerical_eigenforms(61).systems_of_abs(10) # rel tol 1e-9
```

(continues on next page)
systems_of_eigenvalues(bound)

Return all systems of eigenvalues for self for primes up to bound.

EXAMPLES:

```sage
sage: numerical_eigenforms(61).systems_of_eigenvalues(10) # rel tol 1e-9
[[-1.4811943040920152, 0.8060634335253695, 3.1563251746586642, 0.
  →6751308705666477],
  [-1.0, -2.0000000000000027, -3.000000000000003, 1.0000000000000044],
  [0.3111078174659775, 2.903211925911551, 2.525427560843529, 3.214319743377552],
  [2.1700864866626034, 1.7092753594369208, 1.63089761381512, 0.4608112718908984],
  [3.0, 4.0, 6.0, 8.0]]
```

weight()

Return the weight of this set of modular eigenforms.

EXAMPLES:

```sage
sage: n = numerical_eigenforms(61) ; n.weight()
2
```

sage.modular.modform.numerical.support(v, eps)

Given a vector $v$ and a threshold eps, return all indices where $|v|$ is larger than eps.

EXAMPLES:

```sage
sage: sage.modular.modform.numerical.support(numerical_eigenforms(61)._easy_vertex(), 1.0 )
[]
sage: sage.modular.modform.numerical.support(numerical_eigenforms(61)._easy_vertex(), 0.5 )
[0, 4]
```
1.16 The Victor Miller basis

This module contains functions for quick calculation of a basis of $q$-expansions for the space of modular forms of level 1 and any weight. The basis returned is the Victor Miller basis, which is the unique basis of elliptic modular forms $f_1, \ldots, f_d$ for which $a_i(f_j) = \delta_{ij}$ for $1 \leq i, j \leq d$ (where $d$ is the dimension of the space).

This basis is calculated using a standard set of generators for the ring of modular forms, using the fast multiplication algorithms for polynomials and power series provided by the FLINT library. (This is far quicker than using modular symbols).

`sage.modular.modform.vm_basis.delta_qexp(prec=10, var='q', K=Integer Ring)`

Return the $q$-expansion of the weight 12 cusp form $\Delta$ as a power series with coefficients in the ring $K$ (= $\mathbb{Z}$ by default).

INPUT:
- `prec` – integer (default 10), the absolute precision of the output (must be positive)
- `var` – string (default: 'q'), variable name
- `K` – ring (default: $\mathbb{Z}$), base ring of answer

OUTPUT:
a power series over $K$ in the variable `var`

ALGORITHM:
Compute the theta series
\[
\sum_{n \geq 0} (-1)^n (2n + 1) q^{n(n+1)/2},
\]
a very simple explicit modular form whose 8th power is $\Delta$. Then compute the 8th power. All computations are done over $\mathbb{Z}$ or $\mathbb{Z}$ modulo $N$ depending on the characteristic of the given coefficient ring $K$, and coerced into $K$ afterwards.

EXAMPLES:
```
sage: delta_qexp(7)
q - 24*q^2 + 252*q^3 - 1472*q^4 + 4830*q^5 - 6048*q^6 + O(q^7)
sage: delta_qexp(7, 'z')
z - 24*z^2 + 252*z^3 - 1472*z^4 + 4830*z^5 - 6048*z^6 + O(z^7)
sage: delta_qexp(-3)
Traceback (most recent call last):
... ValueError: prec must be positive
```
```
sage: delta_qexp(20, K = GF(3))
q + q^4 + 2*q^7 + 2*q^13 + q^16 + 2*q^19 + O(q^20)
sage: delta_qexp(20, K = GF(3^5, 'a'))
q + q^4 + 2*q^7 + 2*q^13 + q^16 + 2*q^19 + O(q^20)
sage: delta_qexp(10, K = IntegerModRing(60))
q + 36*q^2 + 12*q^3 + 28*q^4 + 30*q^5 + 12*q^6 + 56*q^7 + 57*q^9 + O(q^10)
```

AUTHORS:
- William Stein: original code
- David Harvey (2007-05): sped up first squaring step
- Martin Raum (2009-08-02): use FLINT for polynomial arithmetic (instead of NTL)
sage.modular.modform.vm_basis.victor_miller_basis(k, prec=10, cusp_only=False, var='q')

Compute and return the Victor Miller basis for modular forms of weight $k$ and level 1 to precision $O(q^{\text{prec}})$. If cusp_only is True, return only a basis for the cuspidal subspace.

**INPUT:**

- $k$ – an integer
- prec – (default: 10) a positive integer
- cusp_only – bool (default: False)
- var – string (default: ‘q’)

**OUTPUT:**

A sequence whose entries are power series in $\mathbb{Z}[[\text{var}]]$.

**EXAMPLES:**

```python
sage: victor_miller_basis(1, 6)
[]
sage: victor_miller_basis(0, 6)
[1 + O(q^6)]
sage: victor_miller_basis(2, 6)
[]
sage: victor_miller_basis(4, 6)
[1 + 240*q + 2160*q^2 + 6720*q^3 + 17520*q^4 + 30240*q^5 + O(q^6)]
sage: victor_miller_basis(6, 6, var='w')
[1 - 504*w - 16632*w^2 - 122976*w^3 - 532728*w^4 - 1575504*w^5 + O(w^6)]
sage: victor_miller_basis(6, 6)
[1 - 504*q - 16632*q^2 - 122976*q^3 - 532728*q^4 - 1575504*q^5 + O(q^6)]
sage: victor_miller_basis(12, 6)
[1 + 196560*q^2 + 16773120*q^3 + 398034000*q^4 + 4629381120*q^5 + O(q^6),
q - 24*q^2 + 252*q^3 - 1472*q^4 + 4830*q^5 + O(q^6)]
sage: victor_miller_basis(12, 6, cusp_only=True)
[q - 24*q^2 + 252*q^3 - 1472*q^4 + 4830*q^5 + O(q^6)]
sage: victor_miller_basis(24, 6, cusp_only=True)
[q + 195660*q^3 + 12080128*q^4 + 44656110*q^5 + O(q^6),
q^2 - 48*q^3 + 1080*q^4 - 15040*q^5 + O(q^6)]
```

(continues on next page)
AUTHORS:

• William Stein, Craig Citro: original code
• Martin Raum (2009-08-02): use FLINT for polynomial arithmetic (instead of NTL)

1.17 Compute spaces of half-integral weight modular forms

Based on an algorithm in Basmaj’s thesis.

AUTHORS:

• William Stein (2007-08)

sage.modular.modform.half_integral.half_integral_weight_modform_basis(chi, k, prec)

A basis for the space of weight \( k/2 \) forms with character \( \chi \). The modulus of \( \chi \) must be divisible by 16 and \( k \) must be odd and > 1.

INPUT:

• \( \text{chi} \) – a Dirichlet character with modulus divisible by 16
• \( k \) – an odd integer > 1
• \( \text{prec} \) – a positive integer

OUTPUT: a list of power series

Warning:

1. This code is very slow because it requests computation of a basis of modular forms for integral weight spaces, and that computation is still very slow.
2. If you give an input \( \text{prec} \) that is too small, then the output list of power series may be larger than the dimension of the space of half-integral forms.

EXAMPLES:
We compute some half-integral weight forms of level $16^7$

```sage
half_integral_weight_modform_basis(DirichletGroup(16*7).0^2,3,30)
[q - 2*q^2 - q^9 + 2*q^14 + 6*q^18 - 2*q^21 - 4*q^22 - q^25 + O(q^30),
q^2 - q^14 - 3*q^18 + 2*q^22 + O(q^30),
q^4 - q^8 - q^16 + q^28 + O(q^30),
q^7 - 2*q^15 + O(q^30)]
```

The following illustrates that choosing too low of a precision can give an incorrect answer.

```sage
half_integral_weight_modform_basis(DirichletGroup(16*7).0^2,3,20)
[q - 2*q^2 - q^9 + 2*q^14 + 6*q^18 + O(q^20),
q^2 - q^14 - 3*q^18 + O(q^20),
q^4 - 2*q^8 + 2*q^12 - 4*q^16 + O(q^20),
q^7 - 2*q^8 + 4*q^12 - 2*q^15 - 6*q^16 + O(q^20),
q^8 - 2*q^12 + 3*q^16 + O(q^20)]
```

We compute some spaces of low level and the first few possible weights.

```sage
half_integral_weight_modform_basis(DirichletGroup(16,QQ).1, 3, 10)
[]

half_integral_weight_modform_basis(DirichletGroup(16,QQ).1, 5, 10)
[q - 2*q^3 - 2*q^5 + 4*q^7 - q^9 + O(q^10)]

half_integral_weight_modform_basis(DirichletGroup(16,QQ).1, 7, 10)
[q - 2*q^2 + 4*q^3 + 4*q^4 - 10*q^5 - 16*q^7 + 19*q^9 + O(q^10),
q^2 - 2*q^3 - 2*q^4 + 4*q^5 + 4*q^7 - 8*q^9 + O(q^10),
q^3 - 2*q^5 - 2*q^7 + 4*q^9 + O(q^10)]

half_integral_weight_modform_basis(DirichletGroup(16,QQ).1, 9, 10)
[q - 2*q^2 + 4*q^3 - 8*q^4 + 14*q^5 + 16*q^6 - 40*q^7 + 16*q^8 - 57*q^9 + O(q^10),
q^2 - 2*q^4 + 4*q^5 - 8*q^6 + 20*q^7 - 8*q^8 + 32*q^9 + O(q^10),
q^3 - 2*q^5 - 2*q^7 + 4*q^9 + O(q^10),
q^4 - 2*q^5 - 2*q^6 + 4*q^7 + 4*q^9 + O(q^10),
q^5 - 2*q^7 - 2*q^9 + O(q^10)]
```

This example once raised an error (see github issue #5792).

```sage
half_integral_weight_modform_basis(trivial_character(16),9,10)
[q - 2*q^2 + 4*q^3 - 8*q^4 + 4*q^6 - 16*q^7 + 48*q^8 - 15*q^9 + O(q^10),
q^2 - 2*q^3 + 4*q^4 - 2*q^6 + 8*q^7 - 24*q^8 + O(q^10),
q^3 - 2*q^4 - 4*q^7 + 12*q^8 + O(q^10),
q^4 - 6*q^8 + 0(q^10)]
```


Let $S = S_{k+1}(\epsilon)$ be the space of cusp forms of even integer weight $k + 1$ and character $\epsilon = \chi \psi^{(k+1)/2}$, where $\psi$ is the nontrivial mod-4 Dirichlet character. Let $U$ be the subspace of $S \times S$ of elements $(a, b)$ such that $\Theta_2 a = \Theta_3 b$. Then $U$ is isomorphic to $S_{k/2}(\chi)$ via the map $(a, b) \mapsto a/\Theta_3$. 
1.18 Graded rings of modular forms

This module contains functions to find generators for the graded ring of modular forms of given level.

AUTHORS:

• William Stein (2007-08-24): first version
• David Ayotte (2021-06): implemented category and Parent/Element frameworks

```python
class sage.modular.modform.ring.ModularFormsRing(group, base_ring=Rational Field)
    Bases: Parent

    The ring of modular forms (of weights 0 or at least 2) for a congruence subgroup of SL_2(Z), with coefficients in a specified base ring.

    EXAMPLES:

    sage: ModularFormsRing(Gamma1(13))
    Ring of Modular Forms for Congruence Subgroup Gamma1(13) over Rational Field
    sage: m = ModularFormsRing(4); m
    Ring of Modular Forms for Congruence Subgroup Gamma0(4) over Rational Field
    sage: m.modular_forms_of_weight(2)
    Modular Forms space of dimension 2 for Congruence Subgroup Gamma0(4) of weight 2 over Rational Field
    sage: m.modular_forms_of_weight(10)
    Modular Forms space of dimension 6 for Congruence Subgroup Gamma0(4) of weight 10 over Rational Field
    sage: m == loads(dumps(m))
    True
    sage: m.generators()
    [(2, 1 + 24*q^2 + 24*q^4 + 96*q^6 + 24*q^8 + O(q^10)),
     (2, q + 4*q^3 + 6*q^5 + 8*q^7 + 13*q^9 + O(q^10))]
    sage: m.q_expansion_basis(2, 10)
    [1 + 24*q^2 + 24*q^4 + 96*q^6 + 24*q^8 + O(q^10),
     q + 4*q^3 + 6*q^5 + 8*q^7 + 13*q^9 + O(q^10)]
    sage: m.q_expansion_basis(3, 10)
    []
    sage: m.q_expansion_basis(10, 10)
    [1 + 10560*q^6 + 3960*q^8 + O(q^10),
     q - 8056*q^7 - 30855*q^9 + O(q^10),
     q^2 - 796*q^6 - 8192*q^8 + O(q^10),
     q^3 + 66*q^7 + 832*q^9 + O(q^10),
     q^4 + 40*q^6 + 528*q^8 + O(q^10),
     q^5 + 20*q^7 + 190*q^9 + O(q^10)]
```

Elements of modular forms ring can be initiated via multivariate polynomials (see `from_polynomial()`):

```python
sage: M = ModularFormsRing(1)
sage: M.ngens()
2
sage: E4, E6 = polygens(QQ, 'E4, E6')
sage: M(E4)
1 + 240*q + 2160*q^2 + 6720*q^3 + 17520*q^4 + 30240*q^5 + O(q^6)
sage: M(E6)
1 - 504*q - 16632*q^2 - 122976*q^3 - 532728*q^4 - 1575504*q^5 + O(q^6)
```

(continues on next page)
sage: M((E4^3 - E6^2)/1728)
q - 24*q^2 + 252*q^3 - 1472*q^4 + 4830*q^5 + O(q^6)

Element
alias of GradedModularFormElement

cchange_ring(base_ring)
Return the ring of modular forms over the given base ring and the same group as self.

INPUT:
• base_ring – a base ring, which should be $\mathbb{Q}$, $\mathbb{Z}$, or the integers mod $p$ for some prime $p$.

EXAMPLES:

sage: M = ModularFormsRing(11); M
Ring of Modular Forms for Congruence Subgroup Gamma0(11) over Rational Field
sage: M.change_ring(Zmod(7))
Ring of Modular Forms for Congruence Subgroup Gamma0(11) over Ring of integers modulo 7
sage: M.change_ring(ZZ)
Ring of Modular Forms for Congruence Subgroup Gamma0(11) over Integer Ring

cuspidal_ideal_generators(maxweight=8, prec=None)
Calculate generators for the ideal of cuspidal forms in this ring, as a module over the whole ring.

EXAMPLES:

sage: ModularFormsRing(Gamma0(3)).cuspidal_ideal_generators(maxweight=12)
[(6, q - 6*q^2 + 9*q^3 + 4*q^4 + O(q^5), q - 6*q^2 + 9*q^3 + 4*q^4 + 6*q^5 + O(q^6))]
sage: [k for k,f,F in ModularFormsRing(13, base_ring=ZZ).cuspidal_ideal_generators(maxweight=14)]
[4, 4, 4, 6, 6, 12]

cuspidal_submodule_q_expansion_basis(weight, prec=None)
Calculate a basis of $q$-expansions for the space of cusp forms of weight weight for this group.

INPUT:
• weight (integer) – the weight
• prec (integer or None) – precision of $q$-expansions to return

ALGORITHM: Uses the method cuspidal_ideal_generators() to calculate generators of the ideal of cusp forms inside this ring. Then multiply these up to weight weight using the generators of the whole modular form space returned by q_expansion_basis().

EXAMPLES:

sage: R = ModularFormsRing(Gamma0(3))
sage: R.cuspidal_submodule_q_expansion_basis(20)
[q - 8532*q^6 - 88442*q^7 + O(q^8), q^2 + 207*q^6 + 24516*q^7 + 0(q^8),
q^3 + 456*q^6 + 0(q^8), q^4 - 135*q^6 - 926*q^7 + 0(q^8), q^5 + 18*q^6 + 135*q^7 + O(q^8)]

We compute a basis of a space of very large weight, quickly (using this module) and slowly (using modular symbols), and verify that the answers are the same.
Modular Forms, Release 10.0

sage: A = R.cuspidal_submodule_q_expansion_basis(80, prec=30)  # long time (1s on sage.math, 2013)
sage: B = R.modular_forms_of_weight(80).cuspidal_submodule().q_expansion_basis(prec=30)  # long time (19s on sage.math, 2013)
sage: A == B  # long time
True

from_polynomial(polynomial, gens=None)
Convert the given polynomial to a graded form living in self. If gens is None then the list of generators given by the method gen_forms() will be used. Otherwise, gens should be a list of generators.

INPUT:

• polynomial – A multivariate polynomial. The variables names of the polynomial should be different from 'q'. The number of variable of this polynomial should equal the number of generators

• gens – list (default: None) of generators of the modular forms ring

OUTPUT: A GradedModularFormElement given by the polynomial relation polynomial.

EXAMPLES:

sage: M = ModularFormsRing(1)
sage: x, y = polygens(QQ, 'x, y')
sage: M.from_polynomial(x^2+y^3)
2 - 1032*q + 774072*q^2 - 77047584*q^3 - 11466304584*q^4 - 498052467504*q^5 + O(q^6)
sage: M = ModularFormsRing(Gamma0(6))
sage: M.ngens()
3
sage: x, y, z = polygens(QQ, 'x, y, z')
sage: M.from_polynomial(x+y+z)
1 + q + q^2 + 27*q^3 + q^4 + 6*q^5 + O(q^6)
sage: M.0 + M.1 + M.2
1 + q + q^2 + 27*q^3 + q^4 + 6*q^5 + O(q^6)
sage: P = x.parent()
sage: M.from_polynomial(P(1/2))
1/2

Note that the number of variables must be equal to the number of generators:

sage: x, y = polygens(QQ, 'x, y')
sage: M(x + y)
Traceback (most recent call last):
...
ValueError: the number of variables (2) must be equal to the number of generators of the modular forms ring (3)

gen(i)
Return the i-th generator of self.

INPUT:

• i (Integer) – correspond to the i-th modular form generating the ring of modular forms.

OUTPUT: A GradedModularFormElement

EXAMPLES:
sage: M = ModularFormsRing(1)
sage: E4 = M.0; E4  # indirect doctest
1 + 240*q + 2160*q^2 + 6720*q^3 + 17520*q^4 + 30240*q^5 + O(q^6)
sage: E6 = M.1; E6  # indirect doctest
1 - 504*q - 16632*q^2 - 122976*q^3 - 532728*q^4 - 1575504*q^5 + O(q^6)

\textbf{gen_forms}(\texttt{maxweight=8, start_gens=[], start_weight=2})

Return a list of modular forms generating this ring (as an algebra over the appropriate base ring).

This method differs from \texttt{generators()} only in that it returns graded modular form objects, rather than bare $q$-expansions.

\textbf{INPUT:}

- \texttt{maxweight} (integer, default: 8) – calculate forms generating all forms up to this weight
- \texttt{start_gens} (list, default: [ ]) – a list of modular forms. If this list is nonempty, we find a minimal generating set containing these forms
- \texttt{start_weight} (integer, default: 2) – calculate the graded subalgebra of forms of weight at least \texttt{start_weight}

\textbf{Note:} If called with the default values of \texttt{start_gens} (an empty list) and \texttt{start_weight} (2), the values will be cached for re-use on subsequent calls to this function. (This cache is shared with \texttt{generators()}). If called with non-default values for these parameters, caching will be disabled.

\textbf{EXAMPLES:}

sage: A = ModularFormsRing(Gamma0(11), Zmod(5)).gen_forms(); A
[1 + 12*q^2 + 12*q^3 + 12*q^4 + 12*q^5 + O(q^6),
 q - 2*q^2 - q^3 + 2*q^4 + q^5 + O(q^6),
 q - 9*q^4 - 10*q^5 + O(q^6)]
sage: A[0].parent()
Modular Forms space of dimension 2 for Congruence Subgroup Gamma0(11) of weight 2 over Rational Field

\textbf{generators}(\texttt{maxweight=8, prec=10, start_gens=[], start_weight=2})

If $R$ is the base ring of self, then this function calculates a set of modular forms which generate the $R$-algebra of all modular forms of weight up to \texttt{maxweight} with coefficients in $R$.

\textbf{INPUT:}

- \texttt{maxweight} (integer, default: 8) – check up to this weight for generators
- \texttt{prec} (integer, default: 10) – return $q$-expansions to this precision
- \texttt{start_gens} (list, default: [ ]) – list of pairs $(k, f)$, or triples $(k, f, F)$, where:
  - $k$ is an integer,
  - $f$ is the $q$-expansion of a modular form of weight $k$, as a power series over the base ring of self,
  - $F$ (if provided) is a modular form object corresponding to $F$.

If this list is nonempty, we find a minimal generating set containing these forms. If $F$ is not supplied, then $f$ needs to have sufficiently large precision (an error will be raised if this is not the case); otherwise, more terms will be calculated from the modular form object $F$. 
• **start_weight** (integer, default: 2) – calculate the graded subalgebra of forms of weight at least start_weight.

**OUTPUT:**
a list of pairs \((k, f)\), where \(f\) is the q-expansion to precision prec of a modular form of weight \(k\).

**See also:**
gen_forms(), which does exactly the same thing, but returns Sage modular form objects rather than bare power series, and keeps track of a lifting to characteristic 0 when the base ring is a finite field.

**Note:** If called with the default values of start gens (an empty list) and start_weight (2), the values will be cached for re-use on subsequent calls to this function. (This cache is shared with gen_forms()). If called with non-default values for these parameters, caching will be disabled.

**EXAMPLES:**

```
sage: ModularFormsRing(SL2Z).generators()
[(4, 1 + 240*q + 2160*q^2 + 17520*q^3 + 50200*q^4 + 60480*q^5 + O(q^6)),
 (6, 1 - 504*q + 16632*q^2 - 122976*q^3 + 532728*q^4 - 1575504*q^5 + O(q^6))]
sage: s = ModularFormsRing(SL2Z).generators(maxweight=5, prec=3); s
[(4, 1 + 240*q + 60480*q^5 + O(q^6))]
sage: s[0][1].parent()
Power Series Ring in q over Rational Field
```

Here we see that for \(\Gamma_0(11)\) taking a basis of forms in weights 2 and 4 is enough to generate everything up to weight 12 (and probably everything else):.

```
sage: v = ModularFormsRing(11).generators(maxweight=12)
sage: len(v)
3
sage: [k for k, _ in v]
[2, 2, 4]
sage: from sage.modular.dims import dimension_modular_forms
sage: dimension_modular_forms(11,2)
2
sage: dimension_modular_forms(11,4)
4
```

1.18. Graded rings of modular forms 91
For congruence subgroups not containing \(-1\), we miss out some forms since we can’t calculate weight 1 forms at present, but we can still find generators for the ring of forms of weight $\geq 2$:

```python
sage: ModularFormsRing(Gamma1(4)).generators(prec=10, maxweight=10)
[(2, 1 + 24*q^2 + 24*q^4 + 96*q^6 + 24*q^8 + O(q^10)),
 (2, q + 4*q^3 + 6*q^5 + 8*q^7 + 13*q^9 + O(q^10)),
 (3, 1 + 12*q^2 + 64*q^3 + 60*q^4 + 160*q^6 + 384*q^7 + 252*q^8 + O(q^10)),
 (3, q + 4*q^2 + 8*q^3 + 16*q^4 + 26*q^5 + 32*q^6 + 48*q^7 + 64*q^8 + 73*q^9 + O(q^10))]
```

Using different base rings will change the generators:

```python
sage: ModularFormsRing(Gamma0(13)).generators(maxweight=12, prec=4)
[(2, 1 + 2*q + 6*q^2 + 8*q^3 + O(q^4)),
 (4, 1 + O(q^4)), (4, q + O(q^4)),
 (4, q^2 + 0(q^4)), (4, q^3 + 0(q^4)),
 (6, 1 + O(q^4)),
 (6, q + O(q^4))]
```

```python
sage: ModularFormsRing(Gamma0(13), base_ring=ZZ).generators(maxweight=12, prec=4)
[(2, 1 + 2*q + 6*q^2 + 8*q^3 + O(q^4)),
 (4, q + 4*q^2 + 10*q^3 + O(q^4)),
 (4, 2*q^2 + 5*q^3 + O(q^4)),
 (4, q^2 + O(q^4)),
 (6, O(q^4)),
 (6, O(q^4)),
 (12, O(q^4))]
```

An example where `start_gens` are specified:

```python
sage: M = ModularForms(11, 2); f = (M.0 + M.1).qexp(8)
```

```python
sage: ModularFormsRing(11).generators(start_gens = [(2, f)])
Traceback (most recent call last):
...
ValueError: Requested precision cannot be higher than precision of approximate starting generators!
```

```python
sage: f = (M.0 + M.1).qexp(10); f
1 + 17/5*q + 26/5*q^2 + 43/5*q^3 + 94/5*q^4 + 77/5*q^5 + 154/5*q^6 + 86/5*q^7 +
 36*q^8 + 146/5*q^9 + O(q^10)
```

```python
sage: ModularFormsRing(11).generators(start_gens = [(2, f)])
[(2, 1 + 17/5*q + 26/5*q^2 + 43/5*q^3 + 94/5*q^4 + 77/5*q^5 + 154/5*q^6 + 86/5*q^7 +
 36*q^8 + 146/5*q^9 + O(q^10)),
 (2, 1 + 12*q^2 + 12*q^3 + 12*q^4 + 12*q^5 + 24*q^6 + 24*q^7 + 36*q^8 + 36*q^9 +
 0(q^10)),
 (4, 1 + 0(q^10))]
```

```python
sage: [k for k,f in ModularFormsRing(1, QQ).generators(maxweight=12)]
[4, 6]
```

```python
sage: [k for k,f in ModularFormsRing(1, ZZ).generators(maxweight=12)]
[4, 6, 12]
```

```python
sage: [k for k,f in ModularFormsRing(1, Zmod(5)).generators(maxweight=12)]
[4, 6]
```

```python
sage: [k for k,f in ModularFormsRing(1, Zmod(2)).generators(maxweight=12)]
[4, 6, 12]
```
Return a list of modular forms generating this ring (as an algebra over the appropriate base ring).

This method differs from \texttt{generators()} only in that it returns graded modular form objects, rather than bare \(q\)-expansions.

INPUT:

- \texttt{maxweight} (integer, default: 8) – calculate forms generating all forms up to this weight
- \texttt{start_gens} (list, default: \([\]\)) – a list of modular forms. If this list is nonempty, we find a minimal generating set containing these forms
- \texttt{start_weight} (integer, default: 2) – calculate the graded subalgebra of forms of weight at least \(\texttt{start_weight}\)

\textbf{Note:} If called with the default values of \texttt{start_gens} (an empty list) and \texttt{start_weight} (2), the values will be cached for re-use on subsequent calls to this function. (This cache is shared with \texttt{generators()}). If called with non-default values for these parameters, caching will be disabled.

\begin{verbatim}
sage: A = ModularFormsRing(Gamma0(11), Zmod(5)).gen_forms(); A
[1 + 12*q^2 + 12*q^3 + 12*q^4 + 12*q^5 + O(q^6),
 q - 2*q^2 - q^3 + 2*q^4 + q^5 + O(q^6),
 q - 9*q^4 - 10*q^5 + O(q^6)]
sage: A[0].parent()
Modular Forms space of dimension 2 for Congruence Subgroup Gamma0(11) of weight 2 over Rational Field
\end{verbatim}

\texttt{group()} 
Return the congruence subgroup for which this is the ring of modular forms.

\begin{verbatim}
sage: R = ModularFormsRing(Gamma1(13))
sage: R.group() is Gamma1(13)
True
\end{verbatim}

\texttt{modular_forms_of_weight(weight)}
Return the space of modular forms on this group of the given weight.

\begin{verbatim}
sage: R = ModularFormsRing(13)
sage: R.modular_forms_of_weight(10)
Modular Forms space of dimension 11 for Congruence Subgroup Gamma0(13) of weight 10 over Rational Field
sage: ModularFormsRing(Gamma1(13)).modular_forms_of_weight(3)
Modular Forms space of dimension 20 for Congruence Subgroup Gamma1(13) of weight 3 over Rational Field
\end{verbatim}

\texttt{ngens()} 
Return the number of generators of \texttt{self}

\begin{verbatim}
1.18. Graded rings of modular forms 93
\end{verbatim}
sage: ModularFormsRing(1).ngens()
2
sage: ModularFormsRing(Gamma0(2)).ngens()
2
sage: ModularFormsRing(Gamma1(13)).ngens()  # long time
33

Warning: Computing the number of generators of a graded ring of modular form for a certain congruence subgroup can be very long.

one()

Return the one element of this ring.

EXAMPLES:

sage: M = ModularFormsRing(1)
sage: u = M.one(); u
1
sage: u.is_one()
True
sage: u + u
2
sage: E4 = ModularForms(1,4).0; E4
1 + 240*q + 2160*q^2 + 6720*q^3 + 17520*q^4 + 30240*q^5 + O(q^6)
sage: E4 * u
1 + 240*q + 2160*q^2 + 6720*q^3 + 17520*q^4 + 30240*q^5 + O(q^6)

polynomial_ring(names, gens=None)

Return a polynomial ring of which self is a quotient.

INPUT:

- names – a list or tuple of names (strings), or a comma separated string
- gens (default: None) – (list) a list of generator of self. If gens is None then the generators returned by gen_forms() is used instead.

OUTPUT: A multivariate polynomial ring in the variable names. Each variable of the polynomial ring correspond to a generator given in gens (following the ordering of the list).

EXAMPLES:

sage: M = ModularFormsRing(1)
sage: gens = M.gen_forms()
sage: M.polynomial_ring('E4, E6', gens)
Multivariate Polynomial Ring in E4, E6 over Rational Field
sage: M = ModularFormsRing(Gamma0(8))
sage: gens = M.gen_forms()
sage: M.polynomial_ring('g', gens)
Multivariate Polynomial Ring in g0, g1, g2 over Rational Field

The degrees of the variables are the weights of the corresponding forms:
sage: M = ModularFormsRing(1)
sage: P.<E4, E6> = M.polynomial_ring()
sage: E4.degree()
4
sage: E6.degree()
6
sage: (E4*E6).degree()
10

\textbf{q\_expansion\_basis}(\texttt{weight}, \texttt{prec=None}, \texttt{use\_random=True})

Calculate a basis of q-expansions for the space of modular forms of the given weight for this group, calculated using the ring generators given by \texttt{find\_generators}.

INPUT:

\begin{itemize}
  \item \texttt{weight} (integer) – the weight
  \item \texttt{prec} (integer or \texttt{None}, default: \texttt{None}) – power series precision. If \texttt{None}, the precision defaults to the Sturm bound for the requested level and weight.
  \item \texttt{use\_random} (boolean, default: \texttt{True}) – whether or not to use a randomized algorithm when building up the space of forms at the given weight from known generators of small weight.
\end{itemize}

EXAMPLES:

sage: m = ModularFormsRing(Gamma0(4))
sage: m.q_expansion_basis(2,10)
[1 + 24*q^2 + 24*q^4 + 96*q^6 + 24*q^8 + O(q^10),
 q + 4*q^3 + 6*q^5 + 8*q^7 + 13*q^9 + O(q^10)]
sage: m.q_expansion_basis(3,10)
[]
sage: X = ModularFormsRing(SL2Z)
sage: X.q_expansion_basis(12, 10)
[1 + 196560*q^2 + 16773120*q^3 + 398034000*q^4 + 4629381120*q^5 + 34417656000*q^6 + 187489935360*q^7 + 814879774800*q^8 + 2975551488000*q^9 + O(q^10),
 q - 24*q^2 + 252*q^3 - 1472*q^4 + 4830*q^5 - 6048*q^6 - 113643*q^7 + O(q^8) - 113643*q^9 + O(q^10)]

We calculate a basis of a massive modular forms space, in two ways. Using this module is about twice as fast as Sage’s generic code.

sage: A = ModularFormsRing(11).q_expansion_basis(30, prec=40) # long time (5s)
sage: B = ModularForms(Gamma0(11), 30).q_echelon_basis(prec=40) # long time (9s)
sage: A == B # long time
True

Check that absurdly small values of \texttt{prec} don’t mess things up:

sage: ModularFormsRing(11).q_expansion_basis(10, prec=5)
[1 + O(q^5), q + O(q^5), q^2 + O(q^5), q^3 + O(q^5),
 q^4 + O(q^5), 0(q^5), 0(q^5), 0(q^5), 0(q^5)]

\textbf{some\_elements}()

Return a list of generators of \texttt{self}.

EXAMPLES:
sage: ModularFormsRing(1).some_elements()
[1 + 240*q + 2160*q^2 + 6720*q^3 + 17520*q^4 + 30240*q^5 + O(q^6),
 1 - 504*q - 16632*q^2 - 122976*q^3 - 532728*q^4 - 1575504*q^5 + O(q^6)]

zero()
Return the zero element of this ring.

EXAMPLES:

sage: M = ModularFormsRing(1)
sage: zer = M.zero(); zer
0
sage: zer.is_zero()
True
sage: E4 = ModularForms(1,4).0; E4
1 + 240*q + 2160*q^2 + 6720*q^3 + 17520*q^4 + 30240*q^5 + O(q^6)
sage: E4 + zer
1 + 240*q + 2160*q^2 + 6720*q^3 + 17520*q^4 + 30240*q^5 + O(q^6)
sage: zer * E4
0
sage: E4 * zer
0

1.19 $q$-expansion of $j$-invariant

sage.modular.modform.j_invariant.j_invariant_qexp(prec=10, K=Rational Field)

Return the $q$-expansion of the $j$-invariant to precision $prec$ in the field $K$.

See also:
If you want to evaluate (numerically) the $j$-invariant at certain points, see the special function elliptic_j().

Warning: Stupid algorithm – we divide by Delta, which is slow.

EXAMPLES:

sage: j_invariant_qexp(4)
q^-1 + 744 + 196884*q + 21493760*q^2 + 864299970*q^3 + O(q^4)
sage: j_invariant_qexp(2)
q^-1 + 744 + 196884*q + O(q^2)
sage: j_invariant_qexp(100, GF(2))
q^-1 + q^7 + q^15 + q^31 + q^47 + q^55 + q^71 + q^87 + O(q^100)
1.20 $q$-expansions of theta series

AUTHOR:

• William Stein

sage.modular.modform.theta.theta2_qexp(prec=10, var='q', K=Integer Ring, sparse=False)

Return the $q$-expansion of the series $\theta_2 = \sum_{n \text{ odd}} q^n$.

INPUT:

• prec – integer; the absolute precision of the output
• var – (default: ‘q’) variable name
• K – (default: ZZ) base ring of answer

OUTPUT:

a power series over K

EXAMPLES:

sage: theta2_qexp(18)
q + q^9 + O(q^18)
sage: theta2_qexp(49)
q + q^9 + q^25 + O(q^49)
sage: theta2_qexp(100, 'q', QQ)
q + q^9 + q^25 + q^49 + q^81 + O(q^100)
sage: f = theta2_qexp(100, 't', GF(3)); f
t + t^9 + t^25 + t^49 + t^81 + O(t^100)
sage: parent(f)
Power Series Ring in t over Finite Field of size 3
sage: theta2_qexp(200)
q + q^9 + q^25 + q^49 + q^81 + q^121 + q^169 + O(q^200)
sage: f = theta2_qexp(20, sparse=True); f
q + q^9 + O(q^20)
sage: parent(f)
Sparse Power Series Ring in q over Integer Ring

sage.modular.modform.theta.theta_qexp(prec=10, var='q', K=Integer Ring, sparse=False)

Return the $q$-expansion of the standard $\theta$ series $\theta = 1 + 2 \sum_{n=1}^{\infty} q^n$.

INPUT:

• prec – integer; the absolute precision of the output
• var – (default: ‘q’) variable name
• K – (default: ZZ) base ring of answer

OUTPUT:

a power series over K

EXAMPLES:

sage: theta_qexp(25)
1 + 2*q + 2*q^4 + 2*q^9 + 2*q^16 + O(q^25)
sage: theta_qexp(10)

(continues on next page)
1.21 Design notes

The implementation depends on the fact that we have dimension formulas (see `dims.py`) for spaces of modular forms with character, and new subspaces, so that we don’t have to compute $q$-expansions for the whole space in order to compute $q$-expansions / elements / and dimensions of certain subspaces. Also, the following design is much simpler than the one I used in MAGMA because submodules don’t have lots of complicated special labels. A modular forms module can consist of the span of any elements; they need not be Hecke equivariant or anything else.

The internal basis of $q$-expansions of modular forms for the ambient space is defined as follows:

| First Block: Cuspidal Subspace |
| Second Block: Eisenstein Subspace |
| Cuspidal Subspace: Block for each level `M` dividing `N`, from highest level to lowest. The block for level `M` contains the images at level `N` of the newsubspace of level `M` (basis, then basis(q**d), then basis(q**e), etc.) |
| Eisenstein Subspace: characters, etc. |

Since we can compute dimensions of cuspidal subspaces quickly and easily, it should be easy to locate any of the above blocks. Hence, e.g., to compute basis for new cuspidal subspace, just have to return first $n$ standard basis vector where $n$ is the dimension. However, we can also create completely arbitrary subspaces as well.

The base ring is the ring generated by the character values (or bigger). In MAGMA the base was always $\mathbb{Z}$, which is confusing.
MODULAR FORMS FOR HECKE TRIANGLE GROUPS

2.1 Overview of Hecke triangle groups and modular forms for Hecke triangle groups

AUTHORS:

- Jonas Jermann (2013): initial version

2.1.1 Hecke triangle groups and elements:

- **Hecke triangle group**: The Von Dyck group corresponding to the triangle group with angles \((\pi/2, \pi/n, 0)\) for \(n=3, 4, 5, \ldots\), generated by the conformal circle inversion \(S\) and by the translation \(T\) by \(\lambda = 2 \cos(\pi/n)\). I.e. the subgroup of orientation preserving elements of the triangle group generated by reflections along the boundaries of the above hyperbolic triangle. The group is arithmetic iff \(n=3, 4, 6, \infty\).

The group elements correspond to matrices over \(\mathbb{Z}[\lambda]\), namely the corresponding order in the number field defined by the minimal polynomial of \(\lambda\) (which embeds into \(\text{AlgebraicReal}\) accordingly).

An exact symbolic expression of the corresponding transfinite diameter \(d\) (which is used as a formal parameter for Fourier expansion of modular forms) can be obtained. For arithmetic groups the (correct) rational number is returned instead.

Basic matrices like \(S, T, U, V(j)\) are available.

**EXAMPLES:**

```
sage: from sage.modular.modform_hecketriangle.hecke_triangle_groups import HeckeTriangleGroup
sage: G = HeckeTriangleGroup(12)
sage: G
Hecke triangle group for n = 12
sage: G.is_arithmetic()
False
sage: G.dvalue()
e^(2*euler_gamma - 4*pi/(sqrt(6) + sqrt(2)) + psi(19/24) + psi(17/24))
sage: AA(G.lam())
1.9318516525781...
```

(continues on next page)
• **Decomposition into product of generators:** It is possible to decompose any group element into products of generators the $S$ and $T$. In particular this allows to check whether a given matrix indeed is a group element.

It also allows one to calculate the automorphy factor of a modular form for the Hecke triangle group for arbitrary arguments.

**EXAMPLES:**

```python
sage: from sage.modular.modform_hecketriangle.hecke_triangle_groups import HeckeTriangleGroup
sage: G = HeckeTriangleGroup(6)
sage: G.element_repr_method("basic")
sage: A = G.V(2)*G.V(3)**(-2)
sage: (L, sgn) = A.word_S_T()
sage: L
(S, T**(-2), S, T**(-1), S, T**(-1))
sage: sgn
-1
sage: sgn.parent()  # Hecke triangle group for n = 6
Hecke triangle group for n = 6
```

```python
Traceback (most recent call last):
...
TypeError: The matrix is not an element of Hecke triangle group for n = 6, up to equivalence it identifies two nonequivalent points.
```

```python
sage: G.element_repr_method("basic")
sage: from sage.modular.modform_hecketriangle.space import ModularForms
sage: MF = ModularForms(G, k=4, ep=1)
```
• **Representation of elements:** An element can be represented in several ways:
  - As a matrix over the base ring (default)
  - As a product of the generators S and T
  - As a product of basic blocks conjugated by some element

   EXAMPLES:

   ```sage
   from sage.modular.modform_hecketriangle.hecke_triangle_groups import HeckeTriangleGroup
   G = HeckeTriangleGroup(n=5)
   el = G.S()*G.T(3)*G.S()*G.T(-2)
   ```

   sage: G.element_repr_method("default")
   sage: el
   \[
   \begin{bmatrix}
   -1 & 2*\text{lam} \\
   3*\text{lam} & -6*\text{lam} - 7 
   \end{bmatrix}
   \]
   sage: G.element_repr_method("basic")
   sage: el
   \(S*T^3*S*T^{-2}\)
   sage: G.element_repr_method("block")
   sage: el
   \(-(S*T^3) * (V(4)^2*V(1)^3) * (S*T^3)^{-1}\)
   sage: G.element_repr_method("conj")
   sage: el
   \[-V(4)^2*V(1)^3\]
   sage: G.element_repr_method("default")
   ```

• **Group action on the (extended) upper half plane:** The group action of Hecke triangle groups on the (extended) upper half plane (by linear fractional transformations) is implemented. The implementation is not based on a specific upper half plane model but is defined formally (for arbitrary arguments) instead.

It is possible to determine the group translate of an element in the classic (strict) fundamental domain for the group, together with the corresponding mapping group element.

The corresponding action of the group on itself by conjugation is supported as well.

The usual `slash`-operator for even integer weights is also available. It acts on rational functions (resp. polynomials). For modular forms an evaluation argument is required.

   EXAMPLES:

   ```sage
   from sage.modular.modform_hecketriangle.hecke_triangle_groups import HeckeTriangleGroup
   G = HeckeTriangleGroup(n=7)
   ```

(continues on next page)
sage: G.S().acton(i + exp(-2))
-1/(e^(-2) + I)
sage: A = G.V(2)*G.V(3)^(-2)
sage: A
-S*T^(-2)*S*T^(-1)*S*T^(-1)
sage: A.acton(CC(i + exp(-2)))
0.344549645079... + 0.0163901095115...*I

sage: G.S().acton(A)
-T^(-2)*S*T^(-1)*S*T^(-1)*S
sage: z = AlgebraicField()(4 + 1/7*i)
sage: G.in_FD(z)
False
sage: (A, w) = G.get_FD(z)
sage: A
T^2*S*T^(-1)*S
sage: w
0.516937798396...? + 0.964078044600...?*I
sage: A.acton(w) == z
True
sage: G.in_FD(w)
True
sage: z = PolynomialRing(G.base_ring(), 'z').gen()
sage: rat = z^2 + 1/(z-G.lam())
sage: G.S().slash(rat)
(z^6 - lam*z^4 - z^3)/(-lam*z^4 - z^3)

sage: G.element_repr_method("default")

• Basic properties of group elements: The trace, sign (based on the trace), discriminant and elliptic/parabolic/hyperbolic type are available.

Group elements can be displayed/represented in several ways:

- As matrices over the base ring.
- As a word in (powers of) the generators $S$ and $T$.
- As a word in (powers of) basic block matrices $V(j)$ (resp. $U$, $S$ in the elliptic case) together with the conjugation matrix that maps the element to this form (also see below).

For the case $n=\infty$ the last method is not properly implemented.

EXAMPLES:
• **Fixed points:** Elliptic, parabolic or hyperbolic fixed points of group can be obtained. They are implemented as a (relative) quadratic extension (given by the square root of the discriminant) of the base ring. It is possible to query the correct embedding into a given field.

Note that for hyperbolic (and parabolic) fixed points there is a 1-1 correspondence with primitive hyperbolic/parabolic group elements (at least if \( n < \infty \)). The group action on fixed points resp. on matrices is compatible with this correspondence.

**EXAMPLES:**

```
sage: from sage.modular.modform_hecketriangle.hecke_triangle_groups import HeckeTriangleGroup
sage: G = HeckeTriangleGroup(n=7)

sage: A = G.S()
sage: A.fixed_points()
(1/2*e, -1/2*e)
sage: A.fixed_points(embedded=True)
(I, -I)

sage: A = G.U()
sage: A.fixed_points()
(1/2*e + 1/2*lam, -1/2*e + 1/2*lam)
sage: A.fixed_points(embedded=True)
(0.9009688679024...? + 0.4338837391175...?I, 0.9009688679024...? - 0.4338837391175...

sage: A = -G.V(2)*G.V(3)^(-2)
sage: A.fixed_points()
((-3/7*lam^2 + 2/7*lam + 11/14)*e - 1/7*lam^2 + 3/7*lam + 3/7, (3/7*lam^2 - 2/7*lam -11/14)*e - 1/7*lam^2 + 3/7*lam + 3/7)
sage: A.fixed_points(embedded=True)
(0.3707208390178...?, 1.103231619181...?)
```
sage: el = A.fixed_points()[0]
sage: F = A.root_extension_field()
sage: F == el.parent()
True
sage: A.root_extension_embedding(CC)
Relative number field morphism:
  From: Number Field in e with defining polynomial x^2 - 4*lam^2 - 4*lam + 4 over:
  its base field
  To:  Complex Field with 53 bits of precision
  Defn: e |--> 4.02438434522465
        lam |--> 1.80193773580484
sage: G.V(2).acton(A).fixed_points()[0] == G.V(2).acton(el)
True

• **Lambda-continued fractions:** For parabolic or hyperbolic elements (resp. their corresponding fixed point) the (negative) lambda-continued fraction expansion is eventually periodic. The lambda-CF (i.e. the preperiod and period) is calculated exactly.

In particular this allows to determine primitive and reduced generators of group elements and the corresponding primitive power of the element.

The case \( n=\infty \) is not properly implemented.

**EXAMPLES:**

sage: from sage.modular.modform_hecketriangle.hecke_triangle_groups import HeckeTriangleGroup
sage: G = HeckeTriangleGroup(n=7)
sage: G.element_repr_method("block")
sage: G.V(6).continued_fraction()
((1,), (1, 1, 1, 1, 2))
sage: (-G.V(2)).continued_fraction()
((1,), (2,))

sage: A = -(G.V(2)*G.V(3)^(-2))^2
sage: A.is_primitive()
False
sage: A.primitive_power()
2
sage: A.is_reduced()
False
sage: A.continued_fraction()
((1, 1, 1, 1), (1, 2))

sage: B = A.primitive_part()
sage: B
(-S*T^(-1)*S) * (V(3)) * (-S*T^(-1)*S)^(-1)
sage: B.is_primitive()
True
sage: B.is_reduced()
False
sage: B.continued_fraction()
((1, 1, 1, 1), (1, 2))
sage: A == A.sign() * B^A.primitive_power()
True
sage: B = A.reduce()
sage: B
(T*S*T) * (V(3)) * (T*S*T)^(-1)
sage: B.is_primitive()
True
sage: B.is_reduced()
True
sage: B.continued_fraction()
(((), (1, 2)))
sage: G.element_repr_method("default")

Reduced and simple elements, Hecke-symmetric elements: For primitive conjugacy classes of hyperbolic elements the cycle of reduced elements can be obtain as well as all simple elements. It is also possible to determine whether a class is Hecke-symmetric.

The case \( n = \infty \) is not properly implemented.

EXAMPLES:

sage: from sage.modular.modform_hecketriangle.hecke_triangle_groups import HeckeTriangleGroup
sage: G = HeckeTriangleGroup(n=5)
sage: el = G.V(1)^2*G.V(2)*G.V(4)
sage: R = el.reduced_elements()
sage: [v.continued_fraction() for v in R]
[((), (2, 1, 1, 4)), ((), (1, 1, 4, 2)), ((), (1, 4, 2, 1)), ((), (4, 2, 1, 1))]
sage: el = G.V(1)^2*G.V(2)^2*G.V(4)
sage: R = el.simple_elements()
sage: [v.is_simple() for v in R]
[True, True, True, True]
sage: (fp1, fp2) = R[2].fixed_points(embedded=True)
sage: fp2 < 0 < fp1
True
sage: el = G.V(2)
sage: el.is_hecke_symmetric()
False
sage: (el.simple_fixed_point_set(), el.inverse().simple_fixed_point_set())
((1/2*e, -1/2*lam + 1/2)*e), {-1/2*e, (1/2*lam - 1/2)*e})
sage: el = G.V(2)*G.V(3)
sage: el.is_hecke_symmetric()
True
sage: el.simple_fixed_point_set() == el.inverse().simple_fixed_point_set()
True

Rational period functions: For each primitive (hyperbolic) conjugacy classes and each even weight \( k \) we can
associate a corresponding rational period function. I.e. a rational function \( q \) of weight \( k \) which satisfies: \( q \mid S = 0 \) and \( q + q \mid U + \ldots + q \mid U^{n-1} = 0 \), where \( S, U \) are the corresponding group elements and \( \mid \) is the usual \textit{slash} operator of weight \( k \).

The set of all rational period function is expected to be generated by such functions.

EXAMPLES:

```python
sage: from sage.modular.modform_hecketriangle.hecke_triangle_groups import HeckeTriangleGroup
sage: G = HeckeTriangleGroup(n=5)
sage: S = G.S()
sage: U = G.U()
sage: def is_rpf(f, k=None):
    if not f + S.slash(f, k=k) == 0:
        return False
    if not sum([(U^m).slash(f, k=k) for m in range(G.n())]) == 0:
        return False
    return True

sage: z = PolynomialRing(G.base_ring(), 'z').gen()
sage: [is_rpf(1 - z^(-k), k=k) for k in range(-6, 6, 2)]  # long time
[True, True, True, True, True, True]
sage: [is_rpf(1/z, k=k) for k in range(-6, 6, 2)]
[False, False, False, False, True, False]
sage: el = G.V(2)
sage: el.is_hecke_symmetric()  # False
sage: rpf = el.rational_period_function(-4)
sage: is_rpf(rpf)  # True
sage: rpf
-lam*z^4 + lam
sage: rpf = el.rational_period_function(-2)
sage: is_rpf(rpf)  # True
sage: rpf
(lam + 1)*z^2 - lam - 1
sage: el.rational_period_function(0) == 0  # True
sage: rpf = el.rational_period_function(2)
sage: is_rpf(rpf)  # True
sage: rpf
((lam + 1)*z^2 - lam - 1)/(lam*z^4 + (-lam - 2)*z^2 + lam)

sage: el = G.V(2)*G.V(3)
sage: el.is_hecke_symmetric()  # True
sage: el.rational_period_function(-4) == 0  # True
sage: rpf = el.rational_period_function(-2)
```

(continues on next page)
sage: rpf
(8*lam + 4)*z^2 - 8*lam - 4
sage: rpf = el.rational_period_function(2)
sage: is_rpf(rpf)
True
sage: rpf.denominator()
(144*lam + 89)*z^8 + (-618*lam - 382)*z^6 + (951*lam + 588)*z^4 + (-618*lam -
-382)*z^2 + 144*lam + 89
sage: el.rational_period_function(4) == 0
True
sage: G = HeckeTriangleGroup(n=4)
sage: G.rational_period_functions(k=4, D=12)
[(z^4 - 1)/z^4]
sage: G.rational_period_functions(k=2, D=14)
[(z^2 - 1)/z^2, 1/z, (24*z^6 - 120*z^4 + 120*z^2 - 24)/(9*z^8 - 80*z^6 + 146*z^4 -
-80*z^2 + 9), (24*z^6 - 120*z^4 + 120*z^2 - 24)/(9*z^8 - 80*z^6 + 146*z^4 - 80*z^2,␣
-+ 9)]

• **Block decomposition of elements:** For each group element a very specific conjugacy representative can be obtained. For hyperbolic and parabolic elements the representative is a product \(V(j)\)-matrices. They all have non-negative trace and the number of factors is called the block length of the element (which is implemented).

Note: For this decomposition special care is given to the sign (of the trace) of the matrices.

The case \(n = \infty\) for everything above is not properly implemented.

EXAMPLES:

```python
sage: from sage.modular.modform_hecketriangle.hecke_triangle_groups import HeckeTriangleGroup
sage: G = HeckeTriangleGroup(n=7)
sage: G.element_repr_method("block")
sage: A = -G.V(2)*G.V(6)^3*G.V(3)
sage: A
-(T*S*T) * (V(6)^3*V(3)*V(2)) * (T*S*T)^(-1)
sage: A.sign()
-1
sage: (L, R, sgn) = A.block_decomposition()
sage: L
((-S*T^(-1)*S) * (V(6)^3) * (-S*T^(-1)*S)^(-1), (T*S*T*S*T) * (V(3)) * (T*S*T*S*T)^(-1), (T*S*T) * (V(2)) * (T*S*T)^(-1))
sage: prod(L).sign()
1
sage: A == sgn * (R.acton(prod(L)))
True
sage: t = A.block_length()
sage: t
5
sage: AA(A.discriminant()) >= AA(t^2 * G.lam() - 4)
True
```

• **Class number and class representatives:** The block length provides a lower bound for the discriminant. This
allows to enlist all (representatives of) matrices of (or up to) a given discriminant.

Using the 1-1 correspondence with hyperbolic fixed points (and certain hyperbolic binary quadratic forms) this makes it possible to calculate the corresponding class number (number of conjugacy classes for a given discriminant).

It also allows to list all occurring discriminants up to some bound. Or to enlist all reduced/simple elements resp. their corresponding hyperbolic fixed points for the given discriminant.

Warning: The currently used algorithm is very slow!

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.hecke_triangle_groups import HeckeTriangleGroup
sage: G = HeckeTriangleGroup(n=4)
sage: G.element_repr_method("basic")
sage: G.is_discriminant(68)
True
sage: G.class_number(14)
2
sage: G.list_discriminants(D=68)
[4, 12, 14, 28, 32, 46, 60, 68]
sage: G.list_discriminants(D=0, hyperbolic=False, primitive=False)
[-4, -2, 0]
sage: G.class_number(68)
4
sage: sorted(G.class_representatives(68))
[S*T^(-5)*S*T^(-1)*S, S*T^(-2)*S*T, T*S*T^5, -S*T^(-1)*S*T^2*S*T]
sage: R = G.reduced_elements(68)
sage: all(v.is_reduced() for v in R)  # long time
True
sage: R = G.simple_elements(68)
sage: all(v.is_simple() for v in R)  # long time
True
sage: G.element_repr_method("default")
sage: G = HeckeTriangleGroup(n=5)
sage: G.element_repr_method("basic")
sage: G.list_discriminants(9*G.lam() + 5)
[4*lam, 7*lam + 6, 9*lam + 5]
sage: G.list_discriminants(D=0, hyperbolic=False, primitive=False)
[-4, -lam - 2, lam - 3, 0]
sage: G.class_number(9*G.lam() + 5)
2
sage: sorted(G.class_representatives(9*G.lam() + 5))
[S*T^(-2)*S*T(-1)*S, T*S*T^2]
sage: R = G.reduced_elements(9*G.lam() + 5)
sage: all(v.is_reduced() for v in R)  # long time
True
sage: R = G.simple_elements(9*G.lam() + 5)
sage: for v in R: print(v.string_repr("default"))
[ 1 lam]
[lam + 2  lam]
[lam  1]
```
2.1.2 Modular forms ring and spaces for Hecke triangle groups:

- **Analytic type**: The analytic type of forms, including the behavior at infinity:
  - Meromorphic (and meromorphic at infinity)
  - Weakly holomorphic (holomorphic and meromorphic at infinity)
  - Holomorphic (and holomorphic at infinity)
  - Cuspidal (holomorphic and zero at infinity)

Additionally the type specifies whether the form is modular or only quasi modular.

**EXAMPLES:**

```python
sage: from sage.modular.modform_hecketriangle.analytic_type import AnalyticType
sage: AnalyticType()(["quasi", "cusp"])
quasi cuspidal
```

- **Modular form (for Hecke triangle groups)**: A function of some analytic type which transforms like a modular form for the given group, weight $k$ and multiplier $\epsilon$:
  - $f(z+\lambda) = f(\lambda)$
  - $f(-1/z) = \epsilon * (z/i)^k * f(z)$

  The multiplier is either $1$ or $-1$. The weight is a rational number of the form $4*(n^2+1)/(n-2) + (1-\epsilon)n/(n-2)$. If $n$ is odd, then the multiplier is unique and given by $(-1)^{(k*(n-2)/2)}$. The space of modular forms for a given group, weight and multiplier forms a module over the base ring. It is finite dimensional if the analytic type is holomorphic.

Modular forms can be constructed in several ways:

- Using some already available construction function for modular forms (those function are available for all spaces/rings and in general do not return elements of the same parent)
- Specifying the form as a rational function in the basic generators (see below)
- For weakly holomorphic modular forms it is possible to exactly determine the form by specifying (sufficiently many) initial coefficients of its Fourier expansion.
- There is even hope (no guarantee) to determine a (exact) form from the initial numerical coefficients (see below).
- By specifying the coefficients with respect to a basis of the space (if the corresponding space supports coordinate vectors)
- Arithmetic combination of forms or differential operators applied to forms

The implementation is based on the implementation of the graded ring (see below). All calculations are exact (no precision argument is required). The analytic type of forms is checked during construction. The analytic type of parent spaces after arithmetic/differential operations with elements is changed (extended/reduced) accordingly.

In particular it is possible to multiply arbitrary modular forms (and end up with an element of a modular forms space). If two forms of different weight/multiplier are added then an element of the corresponding modular forms ring is returned instead.
Elements of modular forms spaces are represented by their Fourier expansion.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.space import CuspForms, ModularForms,
    MeromorphicModularForms
sage: MeromorphicModularForms(n=4, k=8, ep=1)
MeromorphicModularForms(n=4, k=8, ep=1) over Integer Ring
sage: CF = CuspForms(n=7, k=12, ep=1)
```

```
sage: MF = ModularForms(k=12, ep=1)
sage: (x,y,z,d) = MF.pol_ring().gens()
Using existing functions:
```
```
sage: CF.Delta()
q + 17/(56*d)*q^2 + 88887/(2458624*d^2)*q^3 + 941331/(481890304*d^3)*q^4 + O(q^5)
```

```
sage: MF(x^3)
1 + 720*q + 179280*q^2 + 16954560*q^3 + 396974160*q^4 + O(q^5)
```

```
sage: qexp = CF.Delta().q_expansion(prec=2)
sage: qexp
q + O(q^2)
sage: qexp.parent()
```

```
Power Series Ring in q over Fraction Field of Univariate Polynomial Ring in d over Integer Ring
```

```
sage: MF(qexp)
q - 24*q^2 + 252*q^3 - 1472*q^4 + O(q^5)
```

```
sage: MF([0,1]) == MF.f_inf()
True
```

```
sage: d = CF.get_d()
sage: MF.E4().serre_derivative() == -1/3 * MF.E6()
True
```

• **Hauptmodul:** The \( j \)-function for Hecke triangle groups is given by the unique Riemann map from the hyperbolic triangle with vertices at \( \rho, i \) and infinity to the upper half plane, normalized such that its Fourier coefficients are real and such that the first nontrivial Fourier coefficient is 1. The function extends to a completely invariant weakly holomorphic function from the upper half plane to the complex numbers. Another used normalization (in capital letters) is \( J(i) = 1 \). The coefficients of \( j \) are rational numbers up to a power of \( d = 1/j(i) \) which is only rational in the arithmetic cases \( n = 3, 4, 6, \infty \).
All Fourier coefficients of modular forms are based on the coefficients of \( j \). The coefficients of \( j \) are calculated by inverting the Fourier series of its inverse (the series inversion is also by far the most expensive operation of all).

**EXAMPLES:**

```python
sage: from sage.modular.modform_hecketriangle.graded_ring import WeakModularFormsRing
sage: from sage.modular.modform_hecketriangle.space import WeakModularForms
sage: WeakModularForms(n=3, k=0, ep=1).j_inv()
q^-1 + 744 + 196884*q + 21493760*q^2 + 864299970*q^3 + 20245856256*q^4 + 0(q^5)
```

• **Basic generators:** There exist unique modular forms \( f_{\rho} \), \( f_i \) and \( f_{\infty} \) such that each has a simple zero at \( \rho=\exp(\pi/n) \), \( i \) and infinity resp. and no other zeros. The forms are normalized such that their first Fourier coefficient is 1. They have the weight and multiplier \((4/(n-2), 1), (2n/(n-2), -1), (4n/(n-2), 1)\) resp. and can be defined in terms of the Hauptmodul \( j \).

**EXAMPLES:**

```python
sage: from sage.modular.modform_hecketriangle.graded_ring import ModularFormsRing
sage: from sage.modular.modform_hecketriangle.space import ModularForms
sage: ModularFormsRing(n=5, red_hom=True).f_rho()
1 + 7/(100*d)*q + 21/(160000*d^2)*q^2 + 1043/(192000000*d^3)*q^3 + 45479/(12288000000000*d^4)*q^4 + O(q^5)
```

• **Eisenstein series and Delta:** The Eisenstein series of weight 2, 4 and 6 exist for all \( n \) and are all implemented. Note that except for \( n=3 \) the series \( E_4 \) and \( E_6 \) do not coincide with \( f_{\rho} \) and \( f_i \). Similarly there always exists a (generalization of) Delta. Except for \( n=3 \) it also does not coincide with \( f_{\infty} \).

In general Eisenstein series of all even weights exist for all \( n \). In the non-arithmetic cases they are however very hard to determine (it’s an open problem?) and consequently not yet implemented, except for trivial one-dimensional cases.

The Eisenstein series in the arithmetic cases \( n = 3, 4, 6 \) are fully implemented though. Note that this requires a lot more work/effort for \( k \neq 2, 4, 6 \) resp. for multidimensional spaces.

The case \( n=\infty \) is a special case (since there are two cusps) and is not implemented yet.

**EXAMPLES:**

```python
sage: from sage.modular.modform_hecketriangle.graded_ring import ModularFormsRing
sage: ModularFormsRing(n=5).E4()
```

(continues on next page)
sage: ModularFormsRing(n=5).E6()
f_rho^2*f_i
sage: ModularFormsRing(n=5).Delta()
f_rho^9*d - f_rho^4*f_i^2*d
sage: ModularFormsRing(n=5).Delta() == ModularFormsRing(n=5).f_˓→inf()^5 * ModularFormsRing(n=5).f_rho()^4
True

The basic generators in some arithmetic cases:

sage: ModularForms(n=3, k=6).E6()
1 - 504*q - 16632*q^2 - 122976*q^3 - 532728*q^4 + O(q^5)
sage: ModularForms(n=4, k=6).E6()
1 - 56q - 2296*q^2 - 13664*q^3 - 73976*q^4 + O(q^5)
sage: ModularForms(n=infinity, k=4).E4()
1 + 16q + 112*q^2 + 448*q^3 + 1136*q^4 + O(q^5)

General Eisenstein series in some arithmetic cases:

sage: ModularFormsRing(n=4).EisensteinSeries(k=8) * 34
25*f_rho^4 + 9*f_i^2
sage: ModularForms(n=3, k=12).EisensteinSeries()
1 + 65526/691*q + 134250480/691*q^2 + 11606736960/691*q^3 + 274945048560/691*q^4 + O(q^5)
sage: ModularForms(n=6, k=12).EisensteinSeries()
1 + 6552/50443*q + 13425048/50443*q^2 + 1165450104/50443*q^3 + 27494504856/50443*q^4 + O(q^5)
sage: ModularForms(n=4, k=22, ep=-1).EisensteinSeries()
1 - 184/53057489*q - 386252984/53057489*q^2 - 1924704989536/53057489*q^3 - 810031218278584/53057489*q^4 + O(q^5)

• Generator for \`k=0\`, \`ep=-1\`: If \(n\) is even then the space of weakly holomorphic modular forms of weight \(0\) and multiplier -1 is not empty and generated by one element, denoted by \(g_{inv}\).

EXAMPLES:

sage: from sage.modular.modform_hecketriangle.space import WeakModularForms
sage: WeakModularForms(n=4, k=0, ep=-1).g_inv()
q^-1 - 24 - 3820*q - 100352*q^2 - 1217598*q^3 - 10797056*q^4 + O(q^5)
sage: WeakModularFormsRing(n=8).g_inv()
(f_rh o^4*f_i)/(f_rh o^8*d - f_i^2*d)

• Quasi modular form (for Hecke triangle groups): \(E2\) no longer transforms like a modular form but like a quasi modular form. More generally quasi modular forms are given in terms of modular forms and powers of \(E2\). E.g. a holomorphic quasi modular form is a sum of holomorphic modular forms multiplied with a power of \(E2\) such that the weights and multipliers match up. The space of quasi modular forms for a given group, weight and multiplier forms a module over the base ring. It is finite dimensional if the analytic type is holomorphic.

The implementation and construction are analogous to modular forms (see above). In particular construction of quasi weakly holomorphic forms by their initial Laurent coefficients is supported as well!

EXAMPLES:
A quasi weak form can be constructed by using its initial Laurent expansion:

```sage
sage: QF = QuasiWeakModularForms(n=8, k=10/3, ep=-1)
sage: qexp = (QF.quasi_part_gens(min_exp=-1)[4]).q_expansion(prec=5)
sage: qexp
q^-1 - 19/(64*d) - 7497/(262144*d^2)*q + 15889/(8388608*d^3)*q^2 + 543834047/(1649267441664*d^4)*q^3 + 711869853/(43980465111040*d^5)*q^4 + O(q^5)
sage: qexp.parent()
Laurent Series Ring in q over Fraction Field of Univariate Polynomial Ring in d over Integer Ring
sage: QF(qexp).as_ring_element()
f_rho^3*f_i*E2^2/(f_rho^8*d - f_i^2*d)
sage: QF(qexp).reduced_parent()
QuasiWeakModularForms(n=8, k=10/3, ep=-1) over Integer Ring
```

Derivatives of (quasi weak) modular forms are again quasi (weak) modular forms:

```sage
sage: CF = QuasiWeakModularForms(n=8, k=10/3, ep=-1)

sage: CF.f_inf().derivative() == CF.f_inf()*CF.E2()
True
```

• **Ring of (quasi) modular forms:** The ring of (quasi) modular forms for a given analytic type and Hecke triangle group. In fact it is a graded algebra over the base ring where the grading is over 1/(n-2)*Z x Z/(2Z) corresponding to the weight and multiplier. A ring element is thus a finite linear combination of (quasi) modular forms of (possibly) varying weights and multipliers.

Each ring element is represented as a rational function in the generators $f_{\rho}$, $f_{i}$ and $E_{2}$. The representations and arithmetic operations are exact (no precision argument is required).

Elements of the ring are represented by the rational function in the generators.

If the parameter `red_hom` is set to `True` (default: `False`) then operations with homogeneous elements try to return an element of the corresponding vector space (if the element is homogeneous) instead of the forms ring. It is also easier to use the forms ring with `red_hom=True` to construct known forms (since then it is not required to specify the weight and multiplier).

**EXAMPLES:**

```sage
sage: from sage.modular.modform_hecketriangle.graded_ring import ModularFormsRing
sage: ModularFormsRing()
ModularFormsRing(n=3) over Integer Ring
sage: (x,y,z,d) = ModularFormsRing().pol_ring().gens()
```

(continues on next page)
• Construction of modular forms spaces and rings: There are functorial constructions behind all forms spaces and rings which assure that arithmetic operations between those spaces and rings work and fit into the coercion framework. In particular ring elements are interpreted as constant modular forms in this context and base extensions are done if necessary.

• Fourier expansion of (quasi) modular forms (for Hecke triangle groups): Each (quasi) modular form (in fact each ring element) possesses a Fourier expansion of the form \( \sum_{n \geq n_0} a_n q^n \), where \( n_0 \) is an integer, \( q = \exp(2\pi i z/\lambda) \) and the coefficients \( a_n \) are rational numbers (or more generally an extension of rational numbers) up to a power of \( d \), where \( d \) is the (possibly) transcendental parameter described above. I.e. the coefficient ring is given by \( \text{Frac}(R)(d) \).

The coefficients are calculated exactly in terms of the (formal) parameter \( d \). The expansion is calculated exactly up to the specified precision. It is also possible to get a Fourier expansion where \( d \) is evaluated to its numerical approximation.

EXAMPLES:

```python
sage: from sage.modular.modform_hecketriangle.graded_ring import ModularFormsRing, QuasiModularFormsRing
sage: ModularFormsRing(n=4).j_inv().q_expansion(prec=3)
q^-1 + 13/(32*d) + 1093/(16384*d^2)*q + 47/(8192*d^3)*q^2 + O(q^3)
sage: QuasiModularFormsRing(n=5).E2().q_expansion(prec=3)
1 - 9/(200*d)*q - 369/(320000*d^2)*q^2 + O(q^3)
sage: QuasiModularFormsRing(n=5).E2().q_expansion_fixed_d(prec=3)
1.00000000000... - 6.380956565426...*q - 23.18584547617...*q^2 + O(q^3)
```

• Evaluation of forms: (Quasi) modular forms (and also ring elements) can be viewed as functions from the upper half plane and can be numerically evaluated by using the Fourier expansion.

The evaluation uses the (quasi) modularity properties (if possible) for a faster and more precise evaluation. The precision of the result depends both on the numerical precision and on the default precision used for the Fourier expansion.

EXAMPLES:

```python
sage: from sage.modular.modform_hecketriangle.graded_ring import ModularFormsRing
sage: f_i = ModularFormsRing(n=4).f_i()
sage: f_i(i)
0
sage: f_i(infinity)
1
sage: f_i(1/7 + 0.01*i)
32189.02016723... + 21226.62951394...*I
```

• L-functions of forms: Using the (pari based) function Dokchitser L-functions of non-constant holomorphic modular forms are supported for all values of \( n \).

Note: For non-arithmetic groups this involves an irrational conductor. The conductor for the arithmetic groups \( n = 3, 4, 6 \), infinity is 1, 2, 3, 4 respectively.
EXAMPLES:

```
sage: from sage.modular.modform.eis_series import eisenstein_series_lseries
sage: from sage.modular.modform_hecketriangle.space import ModularForms
sage: f = ModularForms(n=3, k=4).E4()/240
sage: L = f.lseries()
sage: L.conductor
1
sage: L.check_functional_equation() < 2^(-50)
True
sage: L(1)
-0.0304484570583...

sage: abs(L(1) - eisenstein_series_lseries(4)(1)) < 2^(-53)
True
sage: L.taylor_series(1, 3)
-0.0304484570583... - 0.0504570844798...*z - 0.0350657360354...*z^2 + O(z^3)

sage: coeffs = f.q_expansion_vector(min_exp=0, max_exp=20, fix_d=True)
sage: abs(L(10) - sum([coeffs[k] * ZZ(k)^(-10) for k in range(1,len(coeffs))])
˓→n(53)) < 10^(-7)
True

sage: L = ModularForms(n=6, k=6, ep=-1).E6().lseries(num_prec=200)
sage: L.conductor
3
sage: L.check_functional_equation() < 2^(-180)
True
sage: L.eps
-1

sage: abs(L(3)) < 2^(-180)
True
```

• **(Serre) derivatives:** Derivatives and Serre derivatives of forms can be calculated. The analytic type is extended accordingly.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.graded_ring import ModularFormsRing
sage: from sage.modular.modform_hecketriangle.space import QuasiModularForms
```
(continues on next page)
**Basis for weakly holomorphic modular forms and Faber polynomials:** (Natural) generators of weakly holomorphic modular forms can be obtained using the corresponding generalized Faber polynomials.

**EXAMPLES:**

```python
sage: from sage.modular.modform_hecketriangle.space import WeakModularForms,
     CuspForms
sage: MF = WeakModularForms(n=5, k=62/3, ep=-1)
sage: MF.disp_prec(MF._l1+2)
sage: MF.F_basis(2)
q^2 - 41/(200*d)*q^3 + O(q^4)
sage: MF.F_basis(1)
q - 13071/(64000000*d^2)*q^3 + O(q^4)
sage: MF.F_basis(-0)
1 - 277043/(19200000000000*d^3)*q^3 + O(q^4)
sage: MF.F_basis(-2)
q^-2 - 162727620113/(40960000000000000*d^5)*q^3 + O(q^4)
```

**Basis for quasi weakly holomorphic modular forms:** (Natural) generators of quasi weakly holomorphic modular forms can also be obtained. In most cases it is even possible to find a basis consisting of elements with only one non-trivial Laurent coefficient (up to some coefficient).

**EXAMPLES:**

```python
sage: from sage.modular.modform_hecketriangle.space import QuasiWeakModularForms
sage: QF = QuasiWeakModularForms(n=8, k=10/3, ep=-1)
sage: QF.default_prec(1)
sage: QF.quasi_part_gens(min_exp=-1)
[q^-1 + O(q),
 1 + O(q),
 q^-1 - 9/(128*d) + O(q),
 1 + O(q),
 q^-1 - 19/(64*d) + O(q),
 q^-1 + 1/(64*d) + O(q)]
sage: QF.default_prec(QF.required_laurent_prec(min_exp=-1))
sage: QF.q_basis(min_exp=-1)  # long time
[q^-1 + O(q^5),
 1 + O(q^5),
 q + O(q^5),
 q^2 + O(q^5),
 q^3 + O(q^5),
 q^4 + O(q^5)]
```

**Dimension and basis for holomorphic or cuspidal (quasi) modular forms:** For finite dimensional spaces the dimension and a basis can be obtained.

**EXAMPLES:**
sage: from sage.modular.modform_hecketriangle.space import QuasiModularForms
sage: MF = QuasiModularForms(n=5, k=6, ep=-1)

sage: MF.default_prec(2)
sage: MF.gens()
[1 - 37/(200*d)*q + O(q^2),
 1 + 33/(200*d)*q + O(q^2),
 1 - 27/(200*d)*q + O(q^2)]

Coordinate vectors for (quasi) holomorphic modular forms and (quasi) cusp forms: For (quasi) holomorphic modular forms and (quasi) cusp forms it is possible to determine the coordinate vectors of elements with respect to the basis.

EXAMPLES:

sage: from sage.modular.modform_hecketriangle.space import ModularForms
sage: ModularForms(n=7, k=12, ep=1).dimension()
3
sage: ModularForms(n=7, k=12, ep=1).Delta().coordinate_vector()
(0, 1, 17/(56*d))

sage: from sage.modular.modform_hecketriangle.space import QuasiCuspForms
sage: MF = QuasiCuspForms(n=7, k=20, ep=1)

sage: el = MF(MF.Delta()*MF.E2()^4 + MF.Delta()*MF.E2()*MF.E6())

sage: el.coordinate_vector()
# long time
(0, 0, 0, 1, 29/(196*d), 0, 0, 0, 0, 1, 17/(56*d), 0, 0)

Subspaces: It is possible to construct subspaces of (quasi) holomorphic modular forms or (quasi) cusp forms spaces with respect to a specified basis of the corresponding ambient space. The subspaces also support coordinate vectors with respect to its basis.

EXAMPLES:

sage: from sage.modular.modform_hecketriangle.space import ModularForms
sage: MF = ModularForms(n=7, k=12, ep=1)

sage: subspace = MF.subspace([MF.E4()^3, MF.Delta()])

sage: subspace
Subspace of dimension 2 of ModularForms(n=7, k=12, ep=1) over Integer Ring

sage: el = subspace(MF.E6()^2)

sage: el.coordinate_vector()
(1, -61/(196*d))

sage: el.ambient_coordinate_vector()
(1, -61/(196*d), -51187/(614656*d^2))

sage: from sage.modular.modform_hecketriangle.space import QuasiCuspForms
sage: MF = QuasiCuspForms(n=7, k=20, ep=1)

sage: subspace = MF.subspace([MF.Delta()*MF.E2()^2*MF.E4(), MF.Delta()*MF.E2()^4])

sage: subspace
Subspace of dimension 2 of QuasiCuspForms(n=7, k=20, ep=1) over Integer Ring

sage: el = subspace(MF.Delta()*MF.E2()^4)

# long time

(continues on next page)
sage: el.coordinate_vector()  # long time
(0, 1)
sage: el.ambient_coordinate_vector()  # long time
(0, 0, 0, 0, 0, 0, 0, 1, 17/(56*d), 0, 0)

• Theta subgroup: The Hecke triangle group corresponding to \( n = \infty \) is also completely supported. In particular the (special) behavior around the cusp \(-1\) is considered and can be specified.

EXAMPLES:

sage: from sage.modular.modform.hecketriangle.graded_ring import...
˓→QuasiMeromorphicModularFormsRing
sage: MR = QuasiMeromorphicModularFormsRing(n=\infty, red_hom=True)
sage: MR
QuasiMeromorphicModularFormsRing(n=\infty) over Integer Ring
sage: j_inv = MR.j_inv().full_reduce()
sage: f_i = MR.f_i().full_reduce()
sage: E4 = MR.E4().full_reduce()
sage: E2 = MR.E2().full_reduce()

sage: j_inv
\( q^{-1} + 24 + 276*q + 2048*q^2 + 11202*q^3 + 49152*q^4 + O(q^5) \)
sage: MR.f_rho() == MR(1)
True
sage: E4
1 + 16*q + 112*q^2 + 448*q^3 + 1136*q^4 + O(q^5)
sage: f_i
1 - 24*q + 24*q^2 - 96*q^3 + 24*q^4 + O(q^5)
sage: E2
1 - 8*q - 8*q^2 - 32*q^3 - 40*q^4 + O(q^5)
sage: E4.derivative() == E4 * (E2 - f_i)
True
sage: f_i.serre_derivative() == -1/2 * E4
True
sage: MF = f_i.serre_derivative().parent()
sage: MF
ModularForms(n=\infty, k=4, ep=1) over Integer Ring
sage: MF.dimension()
2
sage: MF.gens()
[1 + 240*q^2 + 2160*q^4 + O(q^5), q - 8*q^2 + 28*q^3 - 64*q^4 + O(q^5)]
sage: E4(i)
1.941017189...

(continues on next page)
1 - \frac{3}{8d}q + O(q^2)

sage: MF.construct_quasi_form(qexp, order_1=-1) == E2/E4
True
sage: MF.disp_prec(6)

sage: MF.q_basis(m=-1, order_1=-1, min_exp=-1)
q^{-1} - \frac{203528}{7}q^5 + O(q^6)

Elements with respect to the full group are automatically coerced to elements of the Theta subgroup if necessary:

sage: el = QuasiMeromorphicModularFormsRing(n=3).Delta().full_reduce() + E2
sage: el
\frac{E4*f_i^4 - 2*E4^2*f_i^2 + E4^3 + 4096*E2}{4096}

sage: el.parent()
QuasiModularFormsRing(n=+Infinity) over Integer Ring

• **Determine exact coefficients from numerical ones:** There is some experimental support for replacing numerical coefficients with corresponding exact coefficients. There is however NO guarantee that the procedure will work (and most probably there are cases where it won’t).

**EXAMPLES:**

```python
sage: from sage.modular.modform_hecketriangle.space import WeakModularForms,
    QuasiCuspForms
sage: WF = WeakModularForms(n=14)

sage: qexp = WF.J_inv().q_expansion_fixed_d(d_num_prec=1000)

sage: qexp.parent()
Laurent Series Ring in q over Real Field with 1000 bits of precision

sage: qexp_int = WF.rationalize_series(qexp)

doctest:...: UserWarning: Using an experimental rationalization of coefficients,
please check the result for correctness!

sage: qexp_int.parent()
Laurent Series Ring in q over Fraction Field of Univariate Polynomial Ring in d

sage: qexp_int == WF.J_inv().q_expansion()
True

sage: qF = QuasiCuspForms(n=8, k=22/3, ep=-1)

sage: el = QF.f_inf()*QF.E2()

sage: qexp = el.q_expansion_fixed_d(d_num_prec=1000)

sage: qexp_int = QF.rationalize_series(qexp)

sage: qexp_int == el.q_expansion()
True

sage: QF(qexp_int) == el
True
```

2.1. Overview of Hecke triangle groups and modular forms for Hecke triangle groups 119
2.1.3 Future ideas:

- Complete support for the case $n=\infty$ (e.g. lambda-CF)
- Properly implemented lambda-CF
- Binary quadratic forms for Hecke triangle groups
- Cycle integrals
- Maybe: Proper spaces (with coordinates) for (quasi) weakly holomorphic forms with bounds on the initial Fourier exponent
- Support for general triangle groups (hard)
- Support for “congruence” subgroups (hard)

2.2 Graded rings of modular forms for Hecke triangle groups

AUTHORS:

- Jonas Jermann (2013): initial version

```python
class sage.modular.modform_hecketriangle.abstract_ring.FormsRing_abstract(group, base_ring, red_hom, n):
    Bases: Parent
    Abstract (Hecke) forms ring.
    This should never be called directly. Instead one should instantiate one of the derived classes of this class.

    AT = Analytic Type
    AnalyticType
        alias of AnalyticType

    Delta()
        Return an analog of the Delta-function.
        It lies in the graded ring of self. In case has_reduce_hom is True it is given as an element of the corresponding space of homogeneous elements.
        It is a cusp form of weight 12 and is equal to $d^3(E4^3 - E6^2)$ or (in terms of the generators) $d^3x^2(2^5n-6)*(x^n - y^2)$.
        Note that Delta is also a cusp form for $n=\infty$.
```

EXAMPLES:

```python
sage: from sage.modular.modform_hecketriangle.graded_ring import ...
    QuasiMeromorphicModularFormsRing, CuspFormsRing
sage: MR = CuspFormsRing(n=7)
sage: Delta = MR.Delta()
sage: Delta in MR
True
sage: Delta
f_rho^15*d - f_rho^8*f_i^2*d
sage: QuasiMeromorphicModularFormsRing(n=7).Delta() == ...
    QuasiMeromorphicModularFormsRing(n=7)(Delta)
```

(continues on next page)
True

```python
sage: from sage.modular.modform_hecketriangle.space import CuspForms,
   ModularForms
sage: MF = CuspForms(n=5, k=12)
sage: Delta = MF.Delta()
sage: Delta in MF
True
sage: CuspFormsRing(n=5, red_hom=True).Delta() == Delta
True
sage: CuspForms(n=5, k=0).Delta() == Delta
True
sage: MF.disp_prec(3)
sage: Delta
q + 47/(200*d)*q^2 + O(q^3)
sage: d = ModularForms(n=5).get_d()
sage: Delta == (d*(ModularForms(n=5).E4()^3-ModularForms(n=5).E6()^2))
True
```

```python
sage: from sage.modular.modform_hecketriangle.series_constructor import MFSeriesConstructor as MFC
sage: MF = CuspForms(n=5, k=12)
sage: d = MF.get_d()
sage: q = MF.get_q()
sage: CuspForms(n=5, k=12).Delta().q_expansion(prec=5) == (d*MFC(group=5, prec=7).Delta_ZZ()(q/d)).add_bigoh(5)
True
sage: CuspForms(n=infinity, k=12).Delta().q_expansion(prec=5) == (d*MFC(group=infinity, prec=7).Delta_ZZ()(q/d)).add_bigoh(5)
True
sage: CuspForms(n=5, k=12).Delta().q_expansion(fix_d=1, prec=5) == MFC(group=5, prec=7).Delta_ZZ().add_bigoh(5)
True
sage: CuspForms(n=infinity, k=12).Delta().q_expansion(fix_d=1, prec=5) == MFC(group=infinity, prec=7).Delta_ZZ().add_bigoh(5)
True
```

```python
sage: CuspForms(n=infinity, k=12).Delta()
q + 24*q^2 + 252*q^3 + 1472*q^4 + O(q^5)
sage: CuspForms(k=12).f_inf() == CuspForms(k=12).Delta()
True
```

```
E2()
Return the normalized quasi holomorphic Eisenstein series of weight 2.
It lies in a (quasi holomorphic) extension of the graded ring of self. In case has_reduce_hom is True it
is given as an element of the corresponding space of homogeneous elements.
It is in particular also a generator of the graded ring of self and the polynomial variable z exactly corre-
```
sponds to $E_2$.

EXAMPLES:

```python
sage: from sage.modular.modform_hecketriangle.graded_ring import
     QuasiMeromorphicModularFormsRing, QuasiModularFormsRing,
     CuspFormsRing
sage: MR = QuasiModularFormsRing(n=7)
sage: E2 = MR.E2()
sage: E2 in MR
True
sage: CuspFormsRing(n=7).E2() == E2
True
sage: E2
E2
sage: QuasiMeromorphicModularFormsRing(n=7).E2() ==
     QuasiMeromorphicModularFormsRing(n=7)(E2)
True
sage: from sage.modular.modform_hecketriangle.space import
     QuasiModularForms, CuspForms
sage: MF = QuasiModularForms(n=5, k=2)
sage: E2 = MF.E2()
sage: E2 in MF
True
sage: QuasiModularFormsRing(n=5, red_hom=True).E2() == E2
True
sage: CuspForms(n=5, k=12, ep=1).E2() == E2
True
sage: MF.disp_prec(3)
sage: E2
1 - 9/(200*d)*q - 369/(320000*d^2)*q^2 + O(q^3)
```

```python
sage: f_inf = MF.f_inf()
sage: E2 == f_inf.derivative() / f_inf
True
```

```python
sage: from sage.modular.modform_hecketriangle.series_constructor import
     MFSeriesConstructor as MFC
sage: MF = QuasiModularForms(n=5, k=2)
sage: d = MF.get_d()
sage: q = MF.get_q()
sage: QuasiModularForms(n=5, k=2).E2().q_expansion(prec=5) == MFC(group=5,
     prec=7).E2_ZZ()(q/d).add_bigoh(5)
True
sage: QuasiModularForms(n=infinity, k=2).E2().q_expansion(prec=5) ==
     MFC(group=infinity, prec=7).E2_ZZ()(q/d).add_bigoh(5)
True
sage: QuasiModularForms(n=5, k=2).E2().q_expansion(fix_d=1, prec=5) ==
     MFC(group=5, prec=7).E2_ZZ().add_bigoh(5)
True
sage: QuasiModularForms(n=infinity, k=2).E2().q_expansion(fix_d=1, prec=5) ==
     MFC(group=infinity, prec=7).E2_ZZ().add_bigoh(5)
True
```

(continues on next page)
sage: QuasiModularForms(n=∞, k=2).E2()
sage: QuasiModularForms(k=2).E2()

E4

Return the normalized Eisenstein series of weight 4.

It lies in a (holomorphic) extension of the graded ring of self. In case has_reduce_hom is True it is given as an element of the corresponding space of homogeneous elements.

It is equal to \( f_{\rho}^{(n-2)} \).

NOTE:

If \( n=∞ \) the situation is different, there we have: \( f_{\rho}=1 \) (since that’s the limit as \( n \) goes to infinity) and the polynomial variable \( x \) refers to \( E4 \) instead of \( f_{\rho} \). In that case \( E4 \) has exactly one simple zero at the cusp \(-1\). Also note that \( E4 \) is the limit of \( f_{\rho}^n \).

EXAMPLES:

```python
sage: from sage.modular.modform_hecketriangle.graded_ring import ...
    QuasiMeromorphicModularFormsRing, ModularFormsRing, CuspFormsRing
sage: MR = ModularFormsRing(n=7)
sage: E4 = MR.E4()
sage: E4 in MR
True
sage: CuspFormsRing(n=7).E4() == E4
True
sage: QuasiMeromorphicModularFormsRing(n=7).E4() == ...
    QuasiMeromorphicModularFormsRing(n=7)(E4)
True

sage: from sage.modular.modform_hecketriangle.space import ModularForms,
    CuspForms
sage: MF = ModularForms(n=5, k=4)
sage: E4 = MF.E4()
sage: E4 in MF
True
sage: ModularFormsRing(n=5, red_hom=True).E4() == E4
True
sage: CuspForms(n=5, k=12).E4() == E4
True
sage: MF.disp_prec(3)
sage: E4
1 + 21/(100*d)*q + 483/(32000*d^2)*q^2 + O(q^3)
```

(continues on next page)
\begin{verbatim}
sage: q = MF.get_q()
sage: ModularForms(n=5, k=4).E4().q_expansion(prec=5) == MFC(group=5, prec=7).E4_ZZ()(q/d).add_bigoh(5)
True
sage: ModularForms(n=infinity, k=4).E4().q_expansion(prec=5) == MFC(group=infinity, prec=7).E4_ZZ()(q/d).add_bigoh(5)
True
sage: ModularForms(n=5, k=4).E4().q_expansion(fix_d=1, prec=5) == MFC(group=5, prec=7).E4_ZZ().add_bigoh(5)
True
sage: ModularForms(n=infinity, k=4).E4().q_expansion(fix_d=1, prec=5) == MFC(group=infinity, prec=7).E4_ZZ().add_bigoh(5)
True
sage: ModularForms(n=infinity, k=4).E4() 1 + 16*q + 112*q^2 + 448*q^3 + 1136*q^4 + O(q^5)
sage: ModularForms(k=4).f_rho() == ModularForms(k=4).E4()
True
sage: ModularForms(k=4).E4() 1 + 240*q + 2160*q^2 + 6720*q^3 + 17520*q^4 + O(q^5)
E6()

Return the normalized Eisenstein series of weight 6.

It lies in a (holomorphic) extension of the graded ring of \texttt{self}. In case \texttt{has\_reduce\_hom} is \texttt{True} it is given as an element of the corresponding space of homogeneous elements.

It is equal to \(f_{\rho}^{n-3} \cdot f_i\).

EXAMPLES:

\begin{verbatim}
sage: from sage.modular.modform_hecketriangle.graded_ring import QuasiMeromorphicModularFormsRing, ModularFormsRing, CuspFormsRing
sage: MR = ModularFormsRing(n=7)
sage: E6 = MR.E6()
sage: E6 in MR
True
sage: CuspFormsRing(n=7).E6() == E6
True
sage: E6
f_rho^4*f_i
sage: QuasiMeromorphicModularFormsRing(n=7).E6() == QuasiMeromorphicModularFormsRing(n=7)(E6)
True
sage: from sage.modular.modform_hecketriangle.space import ModularForms, CuspForms
sage: MF = ModularForms(n=5, k=6)
sage: E6 = MF.E6()
sage: E6 in MF
True
sage: ModularFormsRing(n=5, red_hom=True).E6() == E6
\end{verbatim}
\end{verbatim}
True

\begin{verbatim}
sage: CuspForms(n=5, k=12).E6() == E6
True
sage: MF.disp_prec(3)
sage: E6
1 - 37/(200*d)*q - 14663/(320000*d^2)*q^2 + O(q^3)
\end{verbatim}

\begin{verbatim}
sage: from sage.modular.modform_hecketriangle.series_constructor import MFSeriesConstructor as MFC
sage: MF = ModularForms(n=5, k=6)
sage: d = MF.get_d()
sage: q = MF.get_q()
sage: ModularForms(n=5, k=6).E6().q_expansion(prec=5) == MFC(group=5, prec=7).
˓→E6_ZZ()(q/d).add_bigoh(5)
True
sage: ModularForms(n=infinity, k=6).E6().q_expansion(prec=5) == MFC(group=infinity, prec=7).
˓→E6_ZZ()(q/d).add_bigoh(5)
True
sage: ModularForms(n=5, k=6).E6().q_expansion(fix_d=1, prec=5) == MFC(group=5, ˓→prec=7).E6_ZZ().add_bigoh(5)
True
sage: ModularForms(n=infinity, k=6).E6().q_expansion(fix_d=1, prec=5) == MFC(group=infinity, prec=7).
˓→E6_ZZ().add_bigoh(5)
True
sage: ModularForms(n=infinity, k=6).E6()
1 - 8*q - 248*q^2 - 1952*q^3 - 8440*q^4 + O(q^5)
\end{verbatim}

\begin{verbatim}
sage: ModularForms(k=6).f_i() == ModularForms(k=6).E6()
True
sage: ModularForms(k=6).E6()
1 - 504*q - 16632*q^2 - 122976*q^3 - 532728*q^4 + O(q^5)
\end{verbatim}

\begin{verbatim}
EisensteinSeries(k=None)

Return the normalized Eisenstein series of weight \(k\).

Only arithmetic groups or trivial weights (with corresponding one dimensional spaces) are supported.

INPUT:

\* k – A non-negative even integer, namely the weight.

If \(k=\text{None}\) (default) then the weight of \self\ is choosen if \self\ is homogeneous and the
weight is possible, otherwise \(k=0\) is set.

OUTPUT:

A modular form element lying in a (holomorphic) extension of the graded ring of \self. In case
\has._reduce.hom is True it is given as an element of the corresponding space of homogeneous elements.

EXAMPLES:

\begin{verbatim}
sage: from sage.modular.modform_hecketriangle.graded_ring import ModularFormsRing, CuspFormsRing
sage: MR = ModularFormsRing()
\end{verbatim}

(continues on next page)
sage: MR.EisensteinSeries() == MR.one()
True
sage: E8 = MR.EisensteinSeries(k=8)
sage: E8 in MR
True
sage: E8
\(f_{\rho}^2\)

sage: from sage.modular.modform_hecketriangle.space import CuspForms, ModularForms
sage: MF = ModularForms(n=4, k=12)
sage: E12 = MF.EisensteinSeries()
sage: E12 in MF
True
sage: CuspFormsRing(n=4, red_hom=True).EisensteinSeries(k=12).parent()
ModularForms(n=4, k=12, ep=1) over Integer Ring
sage: MF.disp_prec(4)
sage: E12
1 + 1008/691*q + 2129904/691*q^2 + 178565184/691*q^3 + \(O(q^4)\)

sage: from sage.modular.modform_hecketriangle.series_constructor import MFSeriesConstructor as MFC
sage: d = MF.get_d()
sage: q = MF.get_q()
sage: ModularForms(n=3, k=2).EisensteinSeries().q_expansion(prec=5) == MFC(group=3, prec=7).EisensteinSeries_ZZ(k=2)(q/d).add_bigoh(5)
True
sage: ModularForms(n=3, k=4).EisensteinSeries().q_expansion(prec=5) == MFC(group=3, prec=7).EisensteinSeries_ZZ(k=4)(q/d).add_bigoh(5)
True
sage: ModularForms(n=3, k=6).EisensteinSeries().q_expansion(prec=5) == MFC(group=3, prec=7).EisensteinSeries_ZZ(k=6)(q/d).add_bigoh(5)
True
sage: ModularForms(n=3, k=8).EisensteinSeries().q_expansion(prec=5) == MFC(group=3, prec=7).EisensteinSeries_ZZ(k=8)(q/d).add_bigoh(5)
True
sage: ModularForms(n=4, k=2).EisensteinSeries().q_expansion(prec=5) == MFC(group=4, prec=7).EisensteinSeries_ZZ(k=2)(q/d).add_bigoh(5)
True
sage: ModularForms(n=4, k=4).EisensteinSeries().q_expansion(prec=5) == MFC(group=4, prec=7).EisensteinSeries_ZZ(k=4)(q/d).add_bigoh(5)
True
sage: ModularForms(n=4, k=6).EisensteinSeries().q_expansion(prec=5) == MFC(group=4, prec=7).EisensteinSeries_ZZ(k=6)(q/d).add_bigoh(5)
True
sage: ModularForms(n=4, k=8).EisensteinSeries().q_expansion(prec=5) == MFC(group=4, prec=7).EisensteinSeries_ZZ(k=8)(q/d).add_bigoh(5)
True
sage: ModularForms(n=4, k=2, ep=-1).EisensteinSeries().q_expansion(prec=5) == MFC(group=6, prec=7).EisensteinSeries_ZZ(k=2)(q/d).add_bigoh(5)
True
sage: ModularForms(n=6, k=4).EisensteinSeries().q_expansion(prec=5) ==

(continues on next page)
\[ \text{MFC}(\text{group}=6, \text{prec}=7).\text{EisensteinSeries}_{\mathbb{Z}}(k=4)(q/d).\text{add}_\text{bigoh}(5) \]

True

\[ \text{sage: } \text{ModularForms}(n=6, k=6, \text{ep}=-1).\text{EisensteinSeries}().\text{q}\_\text{expansion}(\text{prec}=5) == \text{MFC}(\text{group}=6, \text{prec}=7).\text{EisensteinSeries}_{\mathbb{Z}}(k=6)(q/d).\text{add}_\text{bigoh}(5) \]

True

\[ \text{sage: } \text{ModularForms}(n=6, k=8).\text{EisensteinSeries}().\text{q}\_\text{expansion}(\text{prec}=5) == \text{MFC}(\text{group}=6, \text{prec}=7).\text{EisensteinSeries}_{\mathbb{Z}}(k=8)(q/d).\text{add}_\text{bigoh}(5) \]

True

\[ \text{sage: } \text{ModularForms}(n=3, k=12).\text{EisensteinSeries}() \]

\[ 1 + \frac{65520}{691}q + \frac{134250480}{691}q^2 + \frac{11606736960}{691}q^3 + \frac{274945048560}{691}q^4 + O(q^5) \]

\[ \text{sage: } \text{ModularForms}(n=4, k=12).\text{EisensteinSeries}() \]

\[ 1 + \frac{1008}{691}q + \frac{2129904}{691}q^2 + \frac{178565184}{691}q^3 + O(q^4) \]

\[ \text{sage: } \text{ModularForms}(n=6, k=12).\text{EisensteinSeries}() \]

\[ 1 + \frac{6552}{50443}q + \frac{13425048}{50443}q^2 + \frac{1165450104}{50443}q^3 + \frac{27494504856}{50443}q^4 + O(q^5) \]

\[ \text{sage: } \text{ModularForms}(n=3, k=20).\text{EisensteinSeries}() \]

\[ 1 + \frac{13200}{174611}q + \frac{6920614800}{174611}q^2 + \frac{15341851377600}{174611}q^3 + \frac{3628395292275600}{174611}q^4 + O(q^5) \]

\[ \text{sage: } \text{ModularForms}(n=4).\text{EisensteinSeries}(k=8) \]

\[ 1 + \frac{480}{17}q + \frac{69600}{17}q^2 + \frac{1050240}{17}q^3 + \frac{8916960}{17}q^4 + O(q^5) \]

\[ \text{sage: } \text{ModularForms}(n=6).\text{EisensteinSeries}(k=20) \]

\[ 1 + \frac{264}{206215591}q + \frac{138412296}{206215591}q^2 + \frac{306852616488}{206215591}q^3 + \frac{72567905845512}{206215591}q^4 + O(q^5) \]

\[
\text{Element}
\text{alias of } FormsRingElement
\]

\[
\text{FormsRingElement}
\text{alias of } FormsRingElement
\]

\[ \text{G}\_\text{inv}(\cdot) \]

If 2 divides \( n \): Return the G-invariant of the group of \( \text{self} \).

The G-invariant is analogous to the J-invariant but has multiplier \( -1 \). I.e. \( \text{G}\_\text{inv}(-1/t) = -\text{G}\_\text{inv}(t) \).

It is a holomorphic square root of \( \text{J}\_\text{inv}^{\ast}(\text{J}\_\text{inv}^{-1}) \) with real Fourier coefficients.

If 2 does not divide \( n \) the function does not exist and an exception is raised.

The G-invariant lies in a (weak) extension of the graded ring of \( \text{self} \). In case \( \text{has}_\text{reduce}_\text{hom} \) is \( \text{True} \) it is given as an element of the corresponding space of homogeneous elements.

NOTE:

If \( n=\text{infinity} \) then \( \text{G}\_\text{inv} \) is holomorphic everywhere except at the cusp \( -1 \) where it isn’t even meromorphic. Consequently this function raises an exception for \( n=\text{infinity} \).

EXAMPLES:

\[ \text{sage: from sage.modular.modform.hecketriangle.graded_ring import } \]

\[ \text{QuasiMeromorphicModularFormsRing, WeakModularFormsRing, CuspFormsRing} \]

\[ \text{sage: MR = WeakModularFormsRing(n=8)} \]

\[ \text{sage: G\_inv = MR.G\_inv()} \]

\[ \text{sage: G\_inv in MR} \]
As explained above, the G-invariant exists only for even `n`:.

```
sage: from sage.modular.modform_hecketriangle.space import WeakModularForms
    WeakModularForms(n=9)
Traceback (most recent call last):
  ... ArithmeticError: G_inv doesn't exist for odd n(=9).
```

**J**: Return the J-invariant (Hauptmodul) of the group of self. It is normalized such that $J(\infty) = \infty$, it has real Fourier coefficients starting with $d > 0$ and $J(i) = 1$.

It lies in a (weak) extension of the graded ring of self. In case has_reduce_hom is True it is given as an

```python
sage: from sage.modular.modform_hecketriangle.space import WeakModularForms
sage: MF = WeakModularForms(n=9)
sage: MF.G_inv()
Traceback (most recent call last):
  ... ArithmeticError: G_inv doesn't exist for odd n(=9).
```
EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.graded_ring import QuasiMeromorphicModularFormsRing, WeakModularFormsRing, CuspFormsRing

sage: MR = WeakModularFormsRing(n=7)
sage: J_inv = MR.J_inv()
sage: J_inv in MR
True

sage: CuspFormsRing(n=7).J_inv() == J_inv
True

sage: f_rho^7/(f_rho^7 - f_i^2)

sage: from sage.modular.modform_hecketriangle.space import WeakModularForms, CuspForms

sage: MF = WeakModularForms(n=5, k=0)
sage: J_inv = MF.J_inv()
sage: J_inv in MF
True

sage: WeakModularFormsRing(n=5, red_hom=True).J_inv() == J_inv
True

sage: CuspForms(n=5, k=12).J_inv() == J_inv
True

sage: MF.disp_prec(3)
sage: J_inv

sage: from sage.modular.modform_hecketriangle.series_constructor import MFSeriesConstructor as MFC

sage: MF = WeakModularForms(n=5)
sage: d = MF.get_d()
sage: q = MF.get_q()
sage: WeakModularForms(n=5).J_inv().q_expansion(prec=5) == MFC(group=5, prec=7).J_inv_ZZ()(q/d).add_bigoh(5)
True

sage: WeakModularForms(n=infinity).J_inv().q_expansion(prec=5) == MFC(group=infinity, prec=7).J_inv_ZZ()(q/d).add_bigoh(5)
True

sage: WeakModularForms(n=infinity).J_inv().q_expansion(fix_d=1, prec=5) == MFC(group=infinity, prec=7).J_inv_ZZ().add_bigoh(5)
True

sage: WeakModularForms(n=infinity).J_inv()
1/64*q^-1 + 3/8 + 69/16*q + 32*q^2 + 5601/32*q^3 + 768*q^4 + O(q^5)

sage: WeakModularForms().J_inv()
```

(continues on next page)
Modular Forms, Release 10.0

(continued from previous page)

\[
1/1728*q^{-1} + 31/72 + 1823/16*q + 335840/27*q^2 + 16005555/32*q^3 + 11716352*q^4 + O(q^5)
\]

**analytic_type()**

Return the analytic type of self.

**EXAMPLES:**

```sage
sage: from sage.modular.modform_hecketriangle.graded_ring import QuasiMeromorphicModularFormsRing, QuasiWeakModularFormsRing
sage: QuasiMeromorphicModularFormsRing().analytic_type()
quasi meromorphic modular
sage: QuasiWeakModularFormsRing().analytic_type()
quasi weakly holomorphic modular
sage: from sage.modular.modform_hecketriangle.space import MeromorphicModularForms, CuspForms
sage: MeromorphicModularForms(k=10).analytic_type()
meromorphic modular
sage: CuspForms(n=7, k=12, base_ring=AA).analytic_type()
cuspidal
```

**base_ring()**

Return base ring of self.

**EXAMPLES:**

```sage
sage: from sage.modular.modform_hecketriangle.graded_ring import ModularFormsRing
sage: ModularFormsRing().base_ring()
Integer Ring
sage: from sage.modular.modform_hecketriangle.space import CuspForms
sage: CuspForms(k=12, base_ring=AA).base_ring()
Algebraic Real Field
```

**change_ring**(new_base_ring)

Return the same space as self but over a new base ring new_base_ring.

**EXAMPLES:**

```sage
sage: from sage.modular.modform_hecketriangle.graded_ring import ModularFormsRing
sage: ModularFormsRing().change_ring(CC)
ModularFormsRing(n=3) over Complex Field with 53 bits of precision
```

**coeff_ring()**

Return coefficient ring of self.

**EXAMPLES:**

```sage
sage: from sage.modular.modform_hecketriangle.graded_ring import ModularFormsRing
sage: ModularFormsRing().coeff_ring()
```
Fraction Field of Univariate Polynomial Ring in d over Integer Ring

```
sage: from sage.modular.modform_hecketriangle.space import CuspForms
sage: CuspForms(k=12, base_ring=AA).coeff_ring()
Fraction Field of Univariate Polynomial Ring in d over Algebraic Real Field
```

**construction()**

Return a functor that constructs self (used by the coercion machinery).

**EXAMPLES:**

```
sage: from sage.modular.modform_hecketriangle.graded_ring import ModularFormsRing
sage: ModularFormsRing().construction()
(ModularFormsRingFunctor(n=3), BaseFacade(Integer Ring))
```

**contains_coeff_ring()**

Return whether self contains its coefficient ring.

**EXAMPLES:**

```
sage: from sage.modular.modform_hecketriangle.graded_ring import CuspFormsRing,
sage: CuspFormsRing(n=4).contains_coeff_ring()
False
sage: ModularFormsRing(n=5).contains_coeff_ring()
True
```

**default_num_prec**(prec=None)

Set the default numerical precision to prec (default: 53). If prec=None (default) the current default numerical precision is returned instead.

**EXAMPLES:**

```
sage: from sage.modular.modform_hecketriangle.space import ModularForms
sage: MF = ModularForms(k=6)
sage: MF.default_prec(20)
sage: MF.default_num_prec(10)
10
sage: E6 = MF.E6()
sage: E6(i + 10^(-1000))
0.002... - 6.7...e-1000*I
```

```
sage: MF.default_num_prec(15)
sage: E6(i + 10^(-1000))
3.9946838...e-1999 - 6.6578064...e-1000*I
```

```
sage: MF = ModularForms(n=5, k=4/3)
sage: f_rho = MF.f_rho()
sage: f_rho.q_expansion(prec=2)[1]
7/(100*d)
```

```
sage: MF.default_num_prec(15)
sage: f_rho.q_expansion_fixed_d(prec=2)[1]
```

(continues on next page)
default_prec(prec=None)

Set the default precision prec for the Fourier expansion. If prec=None (default) then the current default precision is returned instead.

INPUT:

• prec – An integer.

NOTE:

This is also used as the default precision for the Fourier expansion when evaluating forms.

EXAMPLES:

```python
sage: from sage.modular.modform_hecketriangle.graded_ring import ModularFormsRing
sage: MR = ModularFormsRing()
sage: MR.default_prec(3)
sage: MR.default_prec() 3
sage: MR.Delta().q_expansion_fixed_d()
q - 24*q^2 + O(q^3)
```

diff_alg()

Return the algebra of differential operators (over QQ) which is used on rational functions representing elements of self.

EXAMPLES:

```python
sage: ModularFormsRing().diff_alg()
Noncommutative Multivariate Polynomial Ring in X, Y, Z, dX, dY, dZ over Rational Field, nc-relations: {dX*X: X*dX + 1, dY*Y: Y*dY + 1, dZ*Z: Z*dZ + 1}
```

disp_prec(prec=None)

Set the maximal display precision to prec. If prec="max" the precision is set to the default precision. If prec=None (default) then the current display precision is returned instead.
NOTE:

This is used for displaying/representing (elements of) \texttt{self} as Fourier expansions.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.space import ModularForms
sage: MF = ModularForms(k=4)
sage: MF.default_prec(5)
sage: MF.disp_prec(3)
sage: MF.disp_prec()
3
sage: MF.E4()
1 + 240*q + 2160*q^2 + O(q^3)
sage: MF.disp_prec("max")
sage: MF.E4()
1 + 240*q + 2160*q^2 + 6720*q^3 + 17520*q^4 + O(q^5)
```

\texttt{extend_type}(analytic\_type=None, ring=False)

Return a new space which contains (elements of) \texttt{self} with the analytic type of \texttt{self} extended by \texttt{analytic\_type}, possibly extended to a graded ring in case \texttt{ring} is True.

INPUT:

- \texttt{analytic\_type} – An AnalyticType or something which coerces into it (default: None).
- \texttt{ring} – Whether to extend to a graded ring (default: False).

OUTPUT:

The new extended space.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.graded_ring import ModularFormsRing
˓→ ModularFormsRing
sage: from sage.modular.modform_hecketriangle.space import CuspForms
sage: MR = ModularFormsRing(n=5)
sage: MR.extend_type(\"quasi\", \"weak\")
QuasiWeakModularFormsRing(n=5) over Integer Ring
sage: CF=CuspForms(k=12)
sage: CF.extend_type(\"holo\")
ModularForms(n=3, k=12, ep=1) over Integer Ring
sage: CF.extend_type("quasi", \texttt{ring=True})
QuasiCuspFormsRing(n=3) over Integer Ring
sage: CF.subspace([CF.Delta()]).extend_type()
CuspForms(n=3, k=12, ep=1) over Integer Ring
```

\texttt{f\_i}()

Return a normalized modular form \texttt{f\_i} with exactly one simple zero at \texttt{i} (up to the group action).

It lies in a (holomorphic) extension of the graded ring of \texttt{self}. In case \texttt{has\_reduce\_hom} is True it is given as an element of the corresponding space of homogeneous elements.

The polynomial variable \texttt{y} exactly corresponds to \texttt{f\_i}.
EXAMPLES:

```python
sage: from sage.modular.modform_hecketriangle.graded_ring import QuasiMeromorphicModularFormsRing, ModularFormsRing, CuspFormsRing
sage: MR = ModularFormsRing(n=7)
sage: f_i = MR.f_i()
sage: f_i in MR
True
sage: CuspFormsRing(n=7).f_i() == f_i
True
sage: QuasiMeromorphicModularFormsRing(n=7).f_i() == QuasiMeromorphicModularFormsRing(n=7)(f_i)
True
sage: from sage.modular.modform_hecketriangle.space import ModularForms, CuspForms
sage: MF = ModularForms(n=5, k=10/3)
sage: f_i = MF.f_i()
sage: f_i in MF
True
sage: ModularFormsRing(n=5, red_hom=True).f_i() == f_i
True
sage: CuspForms(n=5, k=12).f_i() == f_i
True
sage: MF.disp_prec(3)
sage: f_i
1 - 13/(40*d)*q - 351/(64000*d^2)*q^2 + O(q^3)
```

```python
sage: from sage.modular.modform_hecketriangle.series_constructor import MFSeriesConstructor as MFC
sage: MF = ModularForms(n=5)
sage: d = MF.get_d()
sage: q = MF.get_q()
sage: ModularForms(n=5).f_i().q_expansion(prec=5) == MFC(group=5, prec=7).f_i_ZZ()(q/d).add_bigoh(5)
True
sage: ModularForms(n=infinity).f_i().q_expansion(prec=5) == MFC(group=infinity, prec=7).f_i_ZZ()(q/d).add_bigoh(5)
True
sage: ModularForms(n=5).f_i().q_expansion(fix_d=1, prec=5) == MFC(group=5, prec=7).f_i_ZZ().add_bigoh(5)
True
sage: ModularForms(n=infinity).f_i().q_expansion(fix_d=1, prec=5) == MFC(group=infinity, prec=7).f_i_ZZ().add_bigoh(5)
True
sage: ModularForms(n=infinity, k=2).f_i()
1 - 24*q + 24*q^2 - 96*q^3 + 24*q^4 + O(q^5)
```

```python
sage: ModularForms(k=6).f_i() == ModularForms(k=4).E6()
True
```

(continues on next page)
sage: ModularForms(k=6).f_i()
1 - 504*q - 16632*q^2 - 122976*q^3 - 532728*q^4 + O(q^5)

f_inf()

Return a normalized (according to its first nontrivial Fourier coefficient) cusp form f_inf with exactly one simple zero at infinity (up to the group action).

It lies in a (cuspidal) extension of the graded ring of self. In case has_reduce_hom is True it is given as an element of the corresponding space of homogeneous elements.

NOTE:

If n=infinity then f_inf is no longer a cusp form since it doesn't vanish at the cusp -1. The first nontrivial cusp form is given by E4*f_inf.

EXAMPLES:

sage: from sage.modular.modform_hecketriangle.graded_ring import QuasiMeromorphicModularFormsRing, CuspFormsRing
sage: MR = CuspFormsRing(n=7)
sage: f_inf = MR.f_inf()
sage: f_inf in MR
True
sage: f_inf
f_rho^7*d - f_i^2*d
sage: QuasiMeromorphicModularFormsRing(n=7).f_inf() == QuasiMeromorphicModularFormsRing(n=7)(f_inf)
True
sage: from sage.modular.modform_hecketriangle.space import CuspForms, ModularForms
sage: MF = CuspForms(n=5, k=20/3)
sage: f_inf = MF.f_inf()
sage: f_inf in MF
True
sage: CuspFormsRing(n=5, red_hom=True).f_inf() == f_inf
True
sage: CuspForms(n=5, k=0).f_inf() == f_inf
True
sage: MF.disp_prec(3)
sage: f_inf
q - 9/(200*d)*q^2 + O(q^3)

sage: from sage.modular.modform_hecketriangle.series_constructor import MFSeriesConstructor as MFC
sage: MF = ModularForms(n=5)
sage: d = MF.get_d()
sage: q = MF.get_q()
sage: ModularForms(n=5).f_inf().q_expansion(prec=5) == (d*MFC(group=5, prec=7).f_inf_ZZ()(q/d)).add_bigoh(5)
True
sage: ModularForms(n=infinity).f_inf().q_expansion(prec=5) == (d*MFC(group=infinity, prec=7).f_inf_ZZ()(q/d)).add_bigoh(5)
True

(continues on next page)
f_rho()

Return a normalized modular form f_rho with exactly one simple zero at rho (up to the group action).

It lies in a (holomorphic) extension of the graded ring of self. In case has_reduce_hom is True it is given as an element of the corresponding space of homogeneous elements.

The polynomial variable x exactly corresponds to f_rho.

NOTE:

If n=infinity the situation is different, there we have: f_rho=1 (since that’s the limit as n goes to infinity) and the polynomial variable x no longer refers to f_rho. Instead it refers to E4 which has exactly one simple zero at the cusp -1. Also note that E4 is the limit of f_rho^(n-2).

EXAMPLES:

sage: from sage.modular.modform_hecketriangle.graded_ring import QuasiMeromorphicModularFormsRing, ModularFormsRing, CuspFormsRing
sage: MR = ModularFormsRing(n=7)
sage: f_rho = MR.f_rho()
sage: f_rho in MR
True
sage: CuspFormsRing(n=7).f_rho() == f_rho
True
sage: f_rho

f_rho

sage: QuasiMeromorphicModularFormsRing(n=7).f_rho() == QuasiMeromorphicModularFormsRing(n=7)(f_rho)
True

sage: from sage.modular.modform_hecketriangle.space import ModularForms, CuspForms
sage: MF = ModularForms(n=5, k=4/3)
sage: f_rho = MF.f_rho()
sage: f_rho in MF
True
sage: ModularFormsRing(n=5, red_hom=True).f_rho() == f_rho
True
True

sage: CuspForms(n=5, k=12).f_rho() == f_rho
True

sage: MF.disp_prec(3)
f_rho
1 + 7/(100*d)*q + 21/(160000*d^2)*q^2 + O(q^3)

sage: from sage.modular.modform_hecketriangle.series_constructor import MFSeriesConstructor as MFC
sage: MF = ModularForms(n=5)
sage: d = MF.get_d()
sage: q = MF.get_q()
sage: ModularForms(n=5).f_rho().q_expansion(prec=5) == MFC(group=5, prec=7).f_rho_ZZ()(q/d).add_bigoh(5)
True

sage: ModularForms(n=infinity).f_rho().q_expansion(prec=5) == MFC(group=infinity, prec=7).f_rho_ZZ()(q/d).add_bigoh(5)
True

sage: ModularForms(n=5).f_rho().q_expansion(fix_d=1, prec=5) == MFC(group=5, prec=7).f_rho_ZZ().add_bigoh(5)
True

sage: ModularForms(n=infinity).f_rho().q_expansion(fix_d=1, prec=5) == MFC(group=infinity, prec=7).f_rho_ZZ().add_bigoh(5)
True

sage: ModularForms(n=infinity, k=0).f_rho() == ModularForms(n=infinity, k=0)(1)
True

sage: ModularForms(k=4).f_rho() == ModularForms(k=4).E4()
True

sage: ModularForms(k=4).f_rho()
1 + 240*q + 2160*q^2 + 6720*q^3 + 17520*q^4 + O(q^5)

\( g_inv() \)

If 2 divides \( n \): Return the g-invariant of the group of \( self \).

The g-invariant is analogous to the j-invariant but has multiplier \(-1\). I.e. \( g_inv(-1/t) = -g_inv(t) \). It is a (normalized) holomorphic square root of \( J_inv*(J_inv-1) \), normalized such that its first nontrivial Fourier coefficient is 1.

If 2 does not divide \( n \) the function does not exist and an exception is raised.

The g-invariant lies in a (weak) extension of the graded ring of \( self \). In case \( has_reduced_hom \) is True it is given as an element of the corresponding space of homogeneous elements.

NOTE:

If \( n=\infty \) then \( g_inv \) is holomorphic everywhere except at the cusp \(-1\) where it isn’t even meromorphic. Consequently this function raises an exception for \( n=\infty \).

EXAMPLES:

sage: from sage.modular.modform_hecketriangle.graded_ring import QuasiMeromorphicModularFormsRing, WeakModularFormsRing, CuspFormsRing
sage: MR = WeakModularFormsRing(n=8)

(continues on next page)
As explained above, the \( g \)-invariant exists only for even \( n \)::

```python
sage: from sage.modular.modform_hecketriangle.space import WeakModularForms
sage: MF = WeakModularForms(n=9)
sage: MF.g_inv()
Traceback (most recent call last):
  ...ArithmeticError: g_inv doesn't exist for odd n(=9).
```

**get_d**

Return the parameter \( d \) of self either as a formal parameter or as a numerical approximation with the specified precision (resp. an exact value in the arithmetic cases).

For an (exact) symbolic expression also see \texttt{HeckeTriangleGroup().dvalue()}.

**INPUT:**

- **\texttt{fix\_d}** – If \texttt{False} (default) a formal parameter is used for \( d \).
  
  If \texttt{True} then the numerical value of \( d \) is used (or an exact value if the group is arithmetic). Otherwise, the given value is used for \( d \).

- **\texttt{d\_num\_prec}** – An integer. The numerical precision of \( d \). Default: \texttt{None}, in which case the default numerical precision of \texttt{self.parent()} is used.

**OUTPUT:**
The corresponding formal, numerical or exact parameter $d$ of $\text{self}$, depending on the arguments and whether $\text{self.group()}$ is arithmetic.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.graded_ring import ModularFormsRing
sage: ModularFormsRing(n=8).get_d()
d
sage: ModularFormsRing(n=8).get_d().parent()
Fraction Field of Univariate Polynomial Ring in d over Integer Ring
sage: ModularFormsRing(n=infinity).get_d(fix_d = True)
1/64
sage: ModularFormsRing(n=infinity).get_d(fix_d = True).parent()
Rational Field
sage: ModularFormsRing(n=5).default_num_prec(40)
sage: ModularFormsRing(n=5).get_d(fix_d = True)
0.0070522341...
sage: ModularFormsRing(n=5).get_d(fix_d = True).parent()
Real Field with 40 bits of precision
sage: ModularFormsRing(n=5).get_d(fix_d = True, d_num_prec=100).parent()
Real Field with 100 bits of precision
sage: ModularFormsRing(n=5).get_d(fix_d=1).parent()
Integer Ring
```

\textbf{get\_q}(\text{prec=\text{None}, \text{fix\_d=\text{False}, \text{d\_num\_prec=\text{None}}})

Return the generator of the power series of the Fourier expansion of $\text{self}$.

INPUT:

- \textbf{prec} – An integer or $\text{None}$ (default), namely the desired default precision of the space of power series. If nothing is specified the default precision of $\text{self}$ is used.

- \textbf{fix\_d} – If $\text{False}$ (default) a formal parameter is used for $d$.
  
  If $\text{True}$ then the numerical value of $d$ is used (resp. an exact value if the group is arithmetic). Otherwise the given value is used for $d$.

- \textbf{d\_num\_prec} – The precision to be used if a numerical value for $d$ is substituted.
  
  Default: $\text{None}$ in which case the default numerical precision of $\text{self.parent()}$ is used.

OUTPUT:

The generator of the PowerSeriesRing of corresponding to the given parameters. The base ring of the power series ring is given by the corresponding parent of $\text{self.get\_d()}$ with the same arguments.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.graded_ring import ModularFormsRing
sage: ModularFormsRing(n=8).default_prec(5)
sage: ModularFormsRing(n=8).get_q().parent()
Power Series Ring in q over Fraction Field of Univariate Polynomial Ring in d over Integer Ring
sage: ModularFormsRing(n=8).get_q().parent().default_prec()
5
sage: ModularFormsRing(n=infinity).get_q(prec=12, fix_d = True).parent()
Power Series Ring in q over Rational Field
```

(continues on next page)
graded_ring()

Return the graded ring containing self.

EXAMPLES:

```python
sage: from sage.modular.modform_hecketriangle.graded_ring import *
   ...
   ModularFormsRing, CuspFormsRing
sage: from sage.modular.modform_hecketriangle.space import CuspForms
sage: MR = ModularFormsRing(n=5)
sage: MR.graded_ring() == MR
True
sage: CF=CuspForms(k=12)
sage: CF.graded_ring() == CuspFormsRing()
False
sage: CF.graded_ring() == CuspFormsRing(red_hom=True)
True
sage: CF.subspace([CF.Delta()]).graded_ring() == CuspFormsRing(red_hom=True)
True
```

has_reduce_hom()

Return whether the method reduce should reduce homogeneous elements to the corresponding space of homogeneous elements.

```python
```
This is mainly used by binary operations on homogeneous spaces which temporarily produce an element of self but want to consider it as a homogeneous element (also see reduce).

EXAMPLES:

```python
sage: from sage.modular.modform_hecketriangle.graded_ring import ModularFormsRing
sage: ModularFormsRing().has_reduce_hom()  # False
sage: ModularFormsRing(red_hom=True).has_reduce_hom()  # True

sage: from sage.modular.modform_hecketriangle.space import ModularForms
sage: ModularForms(k=6).has_reduce_hom()  # True
sage: ModularForms(k=6).graded_ring().has_reduce_hom()  # True
```

**hecke_n()**

Return the parameter $n$ of the (Hecke triangle) group of self.

EXAMPLES:

```python
sage: from sage.modular.modform_hecketriangle.graded_ring import ModularFormsRing
sage: MR = ModularFormsRing(n=7)
sage: MR.hecke_n()  # 7

sage: from sage.modular.modform_hecketriangle.space import CuspForms
sage: CF = CuspForms(n=7, k=4/5)
sage: CF.hecke_n()  # 7
```

**homogeneous_part($k$, $ep$)**

Return the homogeneous component of degree ($k$, $e$) of self.

INPUT:

- $k$ – An integer.
- $ep$ – +1 or -1.

EXAMPLES:

```python
sage: from sage.modular.modform_hecketriangle.graded_ring import QuasiMeromorphicModularFormsRing, QuasiWeakModularFormsRing
sage: QuasiMeromorphicModularFormsRing(n=7).homogeneous_part(k=2, ep=-1)  # QuasiMeromorphicModularForms(n=7, k=2, ep=-1) over Integer Ring

```

**is_cuspidal()**

Return whether self only contains cuspidal elements.

EXAMPLES:

```python
sage: from sage.modular.modform_hecketriangle.graded_ring import QuasiModularFormsRing, QuasiCuspFormsRing
```

(continues on next page)
sage: QuasiModularFormsRing().is_cuspidal()
False
sage: QuasiCuspFormsRing().is_cuspidal()
True

sage: from sage.modular.modform_hecketriangle.space import ModularForms,
    QuasiCuspForms
sage: ModularForms(k=12).is_cuspidal()
False
sage: QuasiCuspForms(k=12).is_cuspidal()
True

is_holomorphic()

Return whether self only contains holomorphic modular elements.

EXAMPLES:

sage: from sage.modular.modform_hecketriangle.graded_ring import
    QuasiWeakModularFormsRing, QuasiModularFormsRing
sage: QuasiWeakModularFormsRing().is_holomorphic()
False
sage: QuasiModularFormsRing().is_holomorphic()
True

sage: from sage.modular.modform_hecketriangle.space import WeakModularForms,
    CuspForms
sage: WeakModularForms(k=10).is_holomorphic()
False
sage: CuspForms(n=7, k=12, base_ring=AA).is_holomorphic()
True

is_homogeneous()

Return whether self is homogeneous component.

EXAMPLES:

sage: from sage.modular.modform_hecketriangle.graded_ring import
    ModularFormsRing
sage: ModularFormsRing().is_homogeneous()
False
sage: from sage.modular.modform_hecketriangle.space import ModularForms
sage: ModularForms(k=6).is_homogeneous()
True

is_modular()

Return whether self only contains modular elements.

EXAMPLES:

sage: from sage.modular.modform_hecketriangle.graded_ring import
    QuasiWeakModularFormsRing, CuspFormsRing
sage: QuasiWeakModularFormsRing().is_modular()
False

sage: CuspFormsRing(n=7).is_modular()
True

sage: from sage.modular.modform_hecketriangle.space import QuasiWeakModularForms, CuspForms
sage: QuasiWeakModularForms(k=10).is_modular()
False
sage: CuspForms(n=7, k=12, base_ring=AA).is_modular()
True

is_weakly_holomorphic()

Return whether self only contains weakly holomorphic modular elements.

EXAMPLES:

sage: from sage.modular.modform_hecketriangle.graded_ring import QuasiMeromorphicModularFormsRing, QuasiWeakModularFormsRing, CuspFormsRing
sage: QuasiMeromorphicModularFormsRing().is_weakly_holomorphic()
False
sage: QuasiWeakModularFormsRing().is_weakly_holomorphic()
True
sage: from sage.modular.modform_hecketriangle.space import MeromorphicModularForms, CuspForms
sage: MeromorphicModularForms(k=10).is_weakly_holomorphic()
False
sage: CuspForms(n=7, k=12, base_ring=AA).is_weakly_holomorphic()
True

is_zerospace()

Return whether self is the (0-dimensional) zero space.

EXAMPLES:

sage: from sage.modular.modform_hecketriangle.graded_ring import ModularFormsRing
sage: ModularFormsRing().is_zerospace()
False
sage: from sage.modular.modform_hecketriangle.space import CuspForms
sage: ModularForms(k=12).is_zerospace()
False
sage: CuspForms(k=12).reduce_type([]).is_zerospace()
True

j_inv()

Return the j-invariant (Hauptmodul) of the group of self. It is normalized such that j_inv(infinity) = infinity, and such that it has real Fourier coefficients starting with 1.

It lies in a (weak) extension of the graded ring of self. In case has_reduce_hom is True it is given as an element of the corresponding space of homogeneous elements.

EXAMPLES:
sage: from sage.modular.modform_hecketriangle.graded_ring import QuasiMeromorphicModularFormsRing, WeakModularFormsRing, CuspFormsRing
sage: MR = WeakModularFormsRing(n=7)
sage: j_inv = MR.j_inv()
sage: j_inv in MR
True
sage: j_inv
f_rh0^7/(f_rh0^7*d - f_i^2*d)
sage: QuasiMeromorphicModularFormsRing(n=7).j_inv() == QuasiMeromorphicModularFormsRing(n=7)(j_inv)
True

sage: from sage.modular.modform_hecketriangle.space import WeakModularForms, CuspForms
sage: MF = WeakModularForms(n=5, k=0)
sage: j_inv = MF.j_inv()
sage: j_inv in MF
True
sage: WeakModularFormsRing(n=5, red_hom=True).j_inv() == j_inv
True
sage: CuspForms(n=5, k=12).j_inv() == j_inv
True
sage: MF.disp_prec(3)
sage: j_inv
q^-1 + 79/(200*d) + 42877/(640000*d^2)*q + 12957/(2000000*d^3)*q^2 + O(q^3)

sage: WeakModularForms(n=infinity).j_inv()
q^-1 + 24 + 276*q + 2048*q^2 + 11202*q^3 + 49152*q^4 + O(q^5)

sage: WeakModularForms().j_inv()
q^-1 + 744 + 196884*q + 21493760*q^2 + 86429970*q^3 + 2024586256*q^4 + O(q^5)

pol_ring()
Return the underlying polynomial ring used by self.

EXAMPLES:

rat_field()
Return the underlying rational field used by self to construct/represent elements.

EXAMPLES:
```
sage: from sage.modular.modform_hecketriangle.graded_ring import ModularFormsRing
sage: ModularFormsRing().rat_field()
Fraction Field of Multivariate Polynomial Ring in x, y, z, d over Integer Ring
sage: from sage.modular.modform_hecketriangle.space import CuspForms
sage: CuspForms(k=12, base_ring=AA).rat_field()
Fraction Field of Multivariate Polynomial Ring in x, y, z, d over Algebraic Real Field

reduce_type(\texttt{analytic\_type}=\texttt{None}, \texttt{degree}=\texttt{None})

Return a new space with analytic properties shared by both \texttt{self} and \texttt{analytic\_type}, possibly reduced to its space of homogeneous elements of the given \texttt{degree} (if \texttt{degree} is set). Elements of the new space are contained in \texttt{self}.

INPUT:

\begin{itemize}
\item \texttt{analytic\_type} – An \texttt{AnalyticType} or something which coerces into it (default: \texttt{None}).
\item \texttt{degree} – \texttt{None} (default) or the degree of the homogeneous component to which \texttt{self} should be reduced.
\end{itemize}

OUTPUT:

The new reduced space.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.graded_ring import QuasiModularFormsRing
sage: from sage.modular.modform_hecketriangle.space import QuasiModularForms

sage: MR = QuasiModularFormsRing()
sage: MR.reduce_type(["quasi", "cusp"])
QuasiCuspFormsRing(n=3) over Integer Ring
sage: MR.reduce_type("cusp", degree=(12,1))
CuspForms(n=3, k=12, ep=1) over Integer Ring
sage: MF=QuasiModularForms(k=6)
sage: MF.reduce_type("holo")
ModularForms(n=3, k=6, ep=-1) over Integer Ring
sage: MF.reduce_type([])
ZeroForms(n=3, k=6, ep=-1) over Integer Ring
```

2.2. Graded rings of modular forms for Hecke triangle groups
2.3 Modular forms for Hecke triangle groups

AUTHORS:

- Jonas Jermann (2013): initial version

```python
class sage.modular.modform_hecketriangle.abstract_space.FormsSpace_abstract(group, base_ring, k, ep, n):
    Bases: FormsRing_abstract
    Abstract (Hecke) forms space.
    This should never be called directly. Instead one should instantiate one of the derived classes of this class.
    
    Element
    alias of FormsElement
    
    F_basis(m, order_1=0)
    Returns a weakly holomorphic element of self (extended if necessarily) determined by the property that
    the Fourier expansion is of the form is of the form q^m + O(q^{order_inf + 1}), where order_inf = self._l1 - order_1.
    
    In particular for all m <= order_inf these elements form a basis of the space of weakly holomorphic
    modular forms of the corresponding degree in case n!=infinity.
    
    If n=infinity a non-trivial order of -1 can be specified through the parameter order_1 (default: 0). Otherwise it is ignored.
    
    INPUT:
    
    - m – An integer m <= self._l1.
    - order_1 – The order at -1 of F_simple (default: 0).
      This parameter is ignored if n != infinity.
    
    OUTPUT:
    
    The corresponding element in (possibly an extension of) self. Note that the order at -1 of the resulting
    element may be bigger than order_1 (rare).
    
    EXAMPLES:
```

```python
sage: from sage.modular.modform_hecketriangle.space import WeakModularForms, CuspForms
sage: MF = WeakModularForms(n=5, k=62/3, ep=-1)
sage: MF.disp_prec(MF._l1+2)
sage: MF.weight_parameters()
(2, 3)

sage: MF.F_basis(2)
q^2 - 41/(200*d)*q^3 + O(q^4)
sage: MF.F_basis(1)
q - 13071/(640000*d^2)*q^3 + O(q^4)
sage: MF.F_basis(0)
1 - 277043/(192000000*d^3)*q^3 + O(q^4)
sage: MF.F_basis(-2)
q^-2 - 162727620113/(40960000000000000*d^5)*q^3 + O(q^4)
sage: MF.F_basis(-2).parent() == MF
True
```

(continues on next page)
sage: MF = CuspForms(n=4, k=-2, ep=1)
sage: MF.weight_parameters()
(-1, 3)
sage: MF.F_basis(-1).parent()
WeakModularForms(n=4, k=-2, ep=1) over Integer Ring
sage: MF.F_basis(-1).parent().disp_prec(MF._l1+2)
q^-1 + 80 + O(q)
sage: MF.F_basis(-2)
q^-2 + 400 + O(q)

sage: MF = WeakModularForms(n=infinity, k=14, ep=-1)

sage: MF.F_basis(3)
q^3 - 48*q^4 + O(q^5)
sage: MF.F_basis(2)
q^2 - 1152*q^4 + O(q^5)
sage: MF.F_basis(1)
q - 18496*q^4 + O(q^5)
sage: MF.F_basis(0)
1 - 224280*q^4 + O(q^5)
sage: MF.F_basis(-1)
q^-1 - 2198304*q^4 + O(q^5)

sage: MF.F_basis(3, order_1=-1)
q^3 + O(q^5)
sage: MF.F_basis(1, order_1=2)
q - 300*q^3 - 4096*q^4 + O(q^5)
sage: MF.F_basis(0, order_1=2)
1 - 24*q^2 - 2048*q^3 - 98328*q^4 + O(q^5)
sage: MF.F_basis(-1, order_1=2)
q^-1 - 18150*q^3 - 1327104*q^4 + O(q^5)

F_basis_pol(m, order_1=0)

Returns a polynomial corresponding to the basis element of the corresponding space of weakly holomorphic forms of the same degree as self. The basis element is determined by the property that the Fourier expansion is of the form q^m + O(q^(order_inf + 1)), where order_inf = self._l1 - order_1.

If n=infinity a non-trivial order of -1 can be specified through the parameter order_1 (default: 0). Otherwise it is ignored.

INPUT:

• m – An integer m <= self._l1.
• order_1 – The order at -1 of F_simple (default: 0).
  This parameter is ignored if n != infinity.

OUTPUT:

A polynomial in x,y,z,d, corresponding to f_rho, f_i, E2 and the (possibly) transcendental parameter d.

EXAMPLES:
```python
sage: from sage.modular.modform_hecketriangle.space import WeakModularForms
sage: MF = WeakModularForms(n=5, k=62/3, ep=-1)
sage: MF.weight_parameters()
(2, 3)
sage: MF.F_basis_pol(2)
x^13*y*d^2 - 2*x^8*y^3*d^2 + x^3*y^5*d^2
sage: MF.F_basis_pol(1) * 100
81*x^13*y*d - 62*x^8*y^3*d - 19*x^3*y^5*d
sage: MF.F_basis_pol(0)
(141913*x^13*y + 168974*x^8*y^3 + 9113*x^3*y^5)/320000
sage: MF(MF.F_basis_pol(2)).q_expansion(prec=MF._l1+2)
q^2 - 41/(200*d)*q^3 + O(q^4)
sage: MF(MF.F_basis_pol(1)).q_expansion(prec=MF._l1+1)
q + O(q^3)
sage: MF(MF.F_basis_pol(0)).q_expansion(prec=MF._l1+1)
1 + O(q^3)
sage: MF(MF.F_basis_pol(-2)).q_expansion(prec=MF._l1+1)
q^-2 + O(q^3)
sage: MF(MF.F_basis_pol(-2)).parent()
WeakModularForms(n=5, k=62/3, ep=-1) over Integer Ring
sage: MF = WeakModularForms(n=4, k=-2, ep=1)
sage: MF.weight_parameters()
(-1, 3)
sage: MF.F_basis_pol(-1)
x^3/(x^4*d - y^2*d)
sage: MF.F_basis_pol(-2)
(9*x^7 + 23*x^3*y^2)/(32*x^8*d^2 - 64*x^4*y^2*d^2 + 32*y^4*d^2)
sage: MF(MF.F_basis_pol(-1)).q_expansion(prec=MF._l1+2)
q^-1 + 5/(16*d) + O(q)
sage: MF(MF.F_basis_pol(-2)).q_expansion(prec=MF._l1+2)
q^-2 + 25/(4096*d^2) + O(q)
sage: MF = WeakModularForms(n=infinity, k=14, ep=-1)
sage: MF.F_basis_pol(3)
-y^7*d^3 + 3*x*y^5*d^3 - 3*x^2*y^3*d^3 + x^3*y^3*d^3
sage: MF.F_basis_pol(2)
(3*y^7*d^2 - 17*x*y^5*d^2 + 25*x^2*y^3*d^2 - 11*x^3*y^3*d^2)/(-8)
sage: MF.F_basis_pol(1)
(-75*y^7*d + 225*x*y^5*d - 1249*x^2*y^3*d + 1099*x^3*y^3*d)/1024
sage: MF.F_basis_pol(0)
(41*y^7 - 147*x*y^5 - 1365*x^2*y^3 - 2625*x^3*y)/(-4096)
sage: MF.F_basis_pol(-1)
(-9075*y^9 + 36300*x*y^7 - 718002*x^2*y^5 - 4928052*x^3*y^3 - 2769779*x^4*y)/
(8388608*y^2*d - 8388608*x*d)
sage: MF.F_basis_pol(3, order_1=-1)
(-3*y^9*d^3 + 16*x*y^7*d^3 - 30*x^2*y^5*d^3 + 24*x^3*y^3*d^3 - 7*x^4*y*d^3)/(-
4*x)
```
### Faber_pol

```python
sage: MF.F_basis_pol(1, order_1=2)
-x^2*y^3*d + x^3*y^2*d
sage: MF.F_basis_pol(0, order_1=2)
(-3*x^2*y^3 - 5*x^3*y)/(-8)
sage: MF.F_basis_pol(-1, order_1=2)
(-81*x^2*y^5 - 606*x^3*y^3 - 337*x^4*y)/(1024*y^2*d - 1024*x*d)
```

#### F_simple(order_1=0)

Return a (the most) simple normalized element of `self` corresponding to the weight parameters `l1=self._l1` and `l2=self._l2`. If the element does not lie in `self` the type of its parent is extended accordingly.

The main part of the element is given by the `(l1 - order_1)`-th power of `f_inf`, up to a small holomorphic correction factor.

**INPUT:**

- `order_1` – An integer (default: 0) denoting the desired order at `-1` in the case `n = infinity`. If `n != infinity` the parameter is ignored.

**EXAMPLES:**

```python
sage: from sage.modular.modform_hecketriangle.space import WeakModularForms
sage: MF = WeakModularForms(n=18, k=-7, ep=-1)
sage: MF.disp_prec(1)
sage: MF.F_simple() == MF.f_inf()^MF._l1 * MF.f_rho()^MF._l2 * MF.f_i()
True

sage: from sage.modular.modform_hecketriangle.space import CuspForms,
    ModularForms
sage: MF = CuspForms(n=5, k=2, ep=-1)
sage: MF._l1
-1
sage: MF.F_simple().parent()
WeakModularForms(n=5, k=2, ep=-1) over Integer Ring

sage: MF = ModularForms(n=+Infinity, k=8, ep=1)
sage: MF.F_simple().reduced_parent()
ModularForms(n=+Infinity, k=8, ep=1) over Integer Ring
sage: MF.F_simple(order_1=2)
1 + 32*q + 480*q^2 + 4480*q^3 + 29152*q^4 + O(q^5)
```

#### Faber_pol

Return the `m`th Faber polynomial of `self`.

Namely a polynomial $P(q)$ such that $P(J_inv)*F_simple(order_1)$ has a Fourier expansion of the form $q^m + O(q^{order_inf + 1})$, where `order_inf` = `self._l1` - `order_1` and $d^{(order_inf - m)}*P(q)$ is a monic polynomial of degree `order_inf` - `m`.

If `n=Infinity` a non-trivial order of `-1` can be specified through the parameter `order_1` (default: 0). Otherwise it is ignored.
The Faber polynomials are e.g. used to construct a basis of weakly holomorphic forms and to recover such forms from their initial Fourier coefficients.

INPUT:

- \( \text{m} \) – An integer \( \text{m} \leq \text{order}_{\text{inf}} = \text{self}._{\text{l1}} - \text{order}_1 \).
- \( \text{order}_1 \) – The order at -1 of \( F_{\text{simple}} \) (default: 0).
  This parameter is ignored if \( n \neq \text{infinity} \).
- \( \text{fix}_d \) – If False (default) a formal parameter is used for \( d \).
  If True then the numerical value of \( d \) is used (resp. an exact value if the group is arithmetic).
  Otherwise the given value is used for \( d \).
- \( \text{d_num_prec} \) – The precision to be used if a numerical value for \( d \) is substituted.
  Default: None in which case the default numerical precision of \( \text{self.parent()} \) is used.

OUTPUT:

The corresponding Faber polynomial \( P(q) \).

EXAMPLES:

```python
sage: from sage.modular.modform_hecketriangle.space import WeakModularForms
sage: MF = WeakModularForms(n=5, k=62/3, ep=-1)

sage: MF.weight_parameters()
(2, 3)

sage: MF.Faber_pol(2)
1
sage: MF.Faber_pol(1)
1/d*q - 19/(100*d)
sage: MF.Faber_pol(0)
1/d^2*q^2 - 117/(200*d^2)*q + 9113/(320000*d^2)

sage: MF.Faber_pol(-2)
1/d^4*q^4 - 11/(8*d^4)*q^3 + 41013/(800000*d^4)*q^2 - 2251291/(480000000*d^4)*q + ...
  -> 1974089431/(4915200000000*d^4)

sage: (MF.Faber_pol(2)(MF.J_inv())*MF.F_simple()).q_expansion(prec=MF._l1+2)
q^2 - 41/(200*d)*q^3 + O(q^4)

sage: (MF.Faber_pol(-2)(MF.J_inv())*MF.F_simple()).q_expansion(prec=MF._l1+2, fix_d=1)
q^-2 + O(q^3)
```

(continues on next page)
\[ q^{-2} + O(q^3) \]

```
sage: MF = WeakModularForms(n=4, k=-2, ep=1)
sage: MF.weight_parameters()
(-1, 3)

sage: MF.Faber_pol(-1)
1
sage: MF.Faber_pol(-2, fix_d=True)
256*q - 184
sage: MF.Faber_pol(-3, fix_d=True)
65536*q^2 - 73728*q + 14364
sage: (MF.Faber_pol(-1, fix_d=True)(MF.J_inv())*MF.F_simple()).q_expansion(prec=MF._l1+2, fix_d=True)
q^{-1} + 80 + O(q)
```

```
sage: (MF.Faber_pol(-2, fix_d=True)(MF.J_inv())*MF.F_simple()).q_expansion(prec=MF._l1+2, fix_d=True)
q^{-2} + 400 + O(q)

sage: (MF.Faber_pol(-3)(MF.J_inv())*MF.F_simple()).q_expansion(prec=MF._l1+2, fix_d=True)
q^{-3} + 2240 + O(q)
```

```
sage: MF = WeakModularForms(n=infinity, k=14, ep=-1)
sage: MF.Faber_pol(3)
1
sage: MF.Faber_pol(2)
1/d*q + 3/(8*d)
```

```
sage: MF.Faber_pol(0)
1/d^3*q^3 - 3/(8*d^3)*q^2 + 3/(512*d^3)*q + 41/(4096*d^3)
```

```
sage: MF.Faber_pol(-1)
1/d^4*q^4 - 3/(4*d^4)*q^3 + 81/(1024*d^4)*q^2 + 9075/(8388608*d^4)
```

```
sage: (MF.Faber_pol(-1)(MF.J_inv())*MF.F_simple()).q_expansion(prec=MF._l1 + 1)
q^{-1} - 9075/(8388608*d^4)*q^3 + O(q^4)
```

```
sage: MF.Faber_pol(1, order_1=2)
1
```

```
sage: MF.Faber_pol(0, order_1=2)
1/d*q - 3/(8*d)
```

```
sage: MF.Faber_pol(-1, order_1=2)
1/d^2*q^2 - 3/(4*d^2)*q + 81/(1024*d^2)
```

```
sage: (MF.Faber_pol(-1, order_1=2)(MF.J_inv())*MF.F_simple(order_1=2)).q_expansion(prec=MF._l1 + 1)
q^{-1} - 9075/(8388608*d^4)*q^3 + O(q^4)
```

\textbf{FormsElement}

alias of \texttt{FormsElement}

\texttt{ambient_coordinate_vector(v)}

Return the coordinate vector of the element \( v \) in \texttt{self.module()} with respect to the basis from \texttt{self}.
ambient_space.

NOTE:
Elements use this method (from their parent) to calculate their coordinates.

INPUT:
• v – An element of self.

EXAMPLES:

sage: from sage.modular.modform hecketriangle.space import ModularForms
sage: MF = ModularForms(n=4, k=24, ep=-1)
Vector space of dimension 3 over Fraction Field of Univariate Polynomial Ring
in d over Integer Ring
sage: MF.ambient_coordinate_vector(MF.gen(0))
(1, 0, 0)
sage: subspace = MF.subspace([MF.gen(0), MF.gen(2)])
Vector space of degree 3 and dimension 2 over Fraction Field of Univariate Polynomial Ring in d over Integer Ring
Basis matrix:
[1 0 0]
[0 0 1]
sage: subspace.ambient_coordinate_vector(subspace.gen(0))
(1, 0, 0)

ambient_module()

Return the module associated to the ambient space of self.

EXAMPLES:

sage: from sage.modular.modform hecketriangle.space import ModularForms
sage: MF = ModularForms(k=12)
Vector space of dimension 2 over Fraction Field of Univariate Polynomial Ring
in d over Integer Ring
sage: MF.ambient_module() == MF.module()
True
sage: subspace = MF.subspace([MF.gen(0)])
True

ambient_space()

Return the ambient space of self.

EXAMPLES:


```python
sage: subspace
Subspace of dimension 1 of ModularForms(n=3, k=12, ep=1) over Integer Ring
sage: subspace.ambient_space() == MF
True
```

**aut_factor**(*gamma*, *t*)

The automorphy factor of self.

**INPUT:**

- *gamma* – An element of the group of self.
- *t* – An element of the upper half plane.

**EXAMPLES:**

```python
sage: from sage.modular.modform_hecketriangle.space import ModularForms
sage: MF = ModularForms(n=8, k=4, ep=1)
sage: full_factor = lambda mat, t: (mat[1][0]*t+mat[1][1])**4
sage: T = MF.group().T()
sage: S = MF.group().S()
sage: i = AlgebraicField()(i)
sage: z = 1 + i/2
sage: MF.aut_factor(S, z)
3/2*I - 7/16
sage: MF.aut_factor(-T^(-2), z)
1
sage: MF.aut_factor(MF.group().V(6), z)
173.2640595631...? + 343.8133289126...?*I
sage: MF.aut_factor(S, z) == full_factor(S, z)
True
sage: MF.aut_factor(T, z) == full_factor(T, z)
True
sage: MF.aut_factor(MF.group().V(6), z) == full_factor(MF.group().V(6), z)
True
```

```python
sage: MF = ModularForms(n=7, k=14/5, ep=-1)
sage: T = MF.group().T()
sage: S = MF.group().S()
sage: MF.aut_factor(S, z)
1.3655215324256...? + 0.056805991182877...?*I
sage: MF.aut_factor(-T^(-2), z)
1
sage: MF.aut_factor(S, z) == MF.ep() * (z/i)^MF.weight()
True
sage: MF.aut_factor(MF.group().V(6), z)
13.23058830577...? + 15.71786610686...?*I
```

**change_ring**(*new_base_ring*)

Return the same space as self but over a new base ring *new_base_ring*.

**EXAMPLES:**

2.3. Modular forms for Hecke triangle groups
```python
sage: from sage.modular.modform_hecketriangle.space import CuspForms
sage: CuspForms(n=5, k=24).change_ring(CC)
CuspForms(n=5, k=24, ep=1) over Complex Field with 53 bits of precision
```

**construct_form**(laurent_series, order_1=0, check=True, rationalize=False)

Tries to construct an element of self with the given Fourier expansion. The assumption is made that the specified Fourier expansion corresponds to a weakly holomorphic modular form.

If the precision is too low to determine the element an exception is raised.

**INPUT:**

- **laurent_series** – A Laurent or Power series.
- **order_1** – A lower bound for the order at -1 of the form (default: 0).
  If n\neq\infty this parameter is ignored.
- **check** – If True (default) then the series expansion of the constructed form is compared against the given series.
- **rationalize** – If True (default: False) then the series is rationalized beforehand. Note that in non-exact or non-arithmetic cases this is experimental and extremely unreliable!

**OUTPUT:**

If possible: An element of self with the same initial Fourier expansion as laurent_series.

Note: For modular spaces it is also possible to call self(laurent_series) instead.

**EXAMPLES:**

```python
sage: from sage.modular.modform_hecketriangle.space import CuspForms
sage: Delta = CuspForms(k=12).Delta()
sage: qexp = Delta.q_expansion(prec=2)
sage: qexp.parent()
Power Series Ring in q over Fraction Field of Univariate Polynomial Ring in d over Integer Ring
sage: qexp
q + O(q^2)
sage: CuspForms(k=12).construct_form(qexp) == Delta
True
sage: from sage.modular.modform_hecketriangle.space import WeakModularForms
sage: J_inv = WeakModularForms(n=7).J_inv()
sage: qexp2 = J_inv.q_expansion(prec=1)
sage: qexp2.parent()
Laurent Series Ring in q over Fraction Field of Univariate Polynomial Ring in d over Integer Ring
sage: qexp2
d*q^-1 + 151/392 + O(q)
sage: WeakModularForms(n=7).construct_form(qexp2) == J_inv
True
```

(continues on next page)
(2, 3)
\[
sage: \text{el2} = d \cdot MF.F_basis(2) + 2 \cdot MF.F_basis(1) + MF.F_basis(-2)
\]
\[
sage: \text{qexp2} = \text{el2}.q\_expansion()
\]
\[
sage: \text{qexp2}.parent()
\]
Laurent Series Ring in \( q \) over Fraction Field of Univariate Polynomial Ring in \( d \), \( \longrightarrow \) over Integer Ring
\[
sage: \text{qexp2}
\]
\( q^{-2} + 2q + dq^2 + O(q^3) \)
\[
sage: \text{WeakModularForms}(n=5, k=62/3, \text{ep}=-1).\text{construct(form}(\text{qexp2}) == \text{el2}
\]
True
\[
sage: \text{MF} = \text{WeakModularForms}(n=\infty, k=-2, \text{ep}=-1)
\]
\[
sage: \text{el3} = \text{MF.f_i()}/\text{MF.f_inf()} + \text{MF.f_i()}/\text{MF.f_inf()}/\text{MF.E4()}^2
\]
\[
sage: \text{MF.quasi_part_dimension}(\text{min}\_\text{exp}=-1, \text{order}\_1=-2)
\]
3
\[
sage: \text{prec} = \text{MF}._11 + 3
\]
\[
sage: \text{qexp3} = \text{el3}.q\_expansion(\text{prec})
\]
\[
sage: \text{qexp3}
\]
\( q^{-1} - 1/(4d) + ((1024d^2 - 33)/(1024d^2))*q + O(q^2) \)
\[
sage: \text{MF.construct(form}(\text{qexp3, order}\_1=-2) == \text{el3}
\]
True
\[
sage: \text{MF.construct(form}(\text{el3}.q\_expansion(\text{prec} + 1), \text{order}\_1=-3) == \text{el3}
\]
True
\[
sage: \text{WF} = \text{WeakModularForms}(n=14)
\]
\[
sage: \text{qexp} = \text{WF.J_inv().q_expansion_fixed_d(d_num_prec=1000)}
\]
\[
sage: \text{qexp}.parent()
\]
Laurent Series Ring in \( q \) over Real Field with 1000 bits of precision
\[
sage: \text{WF.construct(form}(\text{qexp, rationalize=True}) == \text{WF.J_inv()}
\]
doctest:...: UserWarning: Using an experimental rationalization of coefficients, please check the result for correctness!
True

**construct_quasi_form**(laurent_series, order_1=0, check=True, rationalize=False)

Try to construct an element of self with the given Fourier expansion. The assumption is made that the specified Fourier expansion corresponds to a weakly holomorphic quasi modular form.

If the precision is too low to determine the element an exception is raised.

**INPUT:**

- **laurent_series** – A Laurent or Power series.
- **order_1** – A lower bound for the order at \(-1\) for all quasi parts of the form (default: 0). If \(n!=\infty\) this parameter is ignored.
- **check** – If True (default) then the series expansion of the constructed form is compared against the given (rationalized) series.
- **rationalize** – If True (default: False) then the series is rationalized beforehand. Note that in non-exact or non-arithmetic cases this is experimental and extremely unreliable!

**OUTPUT:**

If possible: An element of self with the same initial Fourier expansion as \(\text{laurent_series}\.\)
Note: For non modular spaces it is also possible to call `self(laurent_series)` instead. Also note that this function works much faster if a corresponding (cached) `q_basis` is available.

**EXAMPLES:**

```python
sage: from sage.modular.modform_hecketriangle.space import _QuasiWeakModularForms, ModularForms, QuasiModularForms, QuasiCuspForms
sage: QF = _QuasiWeakModularForms(n=8, k=10/3, ep=-1)
```

```python
e1 = QF.quasi_part_gens(min_exp=-1)[4]
sage: prec = QF.required_laurent_prec(min_exp=-1)
sage: prec
5
sage: qexp = e1.q_expansion(prec=prec)
sage: qexp
q^-1 - 19/(64*d) - 7497/(262144*d^2)*q + 15889/(8388608*d^3)*q^2 + 543834047/(1649267441664*d^4)*q^3 + 711869853/(4398046511040*d^5)*q^4 + O(q^5)
sage: qexp.parent()
Laurent Series Ring in q over Fraction Field of Univariate Polynomial Ring in d
over Integer Ring
```

```python
sage: constructed_e1 = QF.construct_quasi_form(qexp)
sage: constructed_e1.parent()
QuasiWeakModularForms(n=8, k=10/3, ep=-1) over Integer Ring
sage: e1 == constructed_e1
True
```

If a `q_basis` is available the construction uses a different algorithm which we also check:

```python
sage: basis = QF.q_basis(min_exp=-1)
sage: QF(qexp) == constructed_e1
True
```

```python
sage: MF = ModularForms(k=36)
sage: e1 = MF.quasi_part_gens(min_exp=2)[1]
sage: prec = MF.required_laurent_prec(min_exp=2)
sage: prec
4
sage: qexp2 = e1.q_expansion(prec=prec + 1)
sage: qexp2
q^3 - 1/(24*d)*q^4 + O(q^5)
sage: qexp2.parent()
Power Series Ring in q over Fraction Field of Univariate Polynomial Ring in d
over Integer Ring
```

```python
sage: constructed_e12 = MF.construct_quasi_form(qexp2)
sage: constructed_e12.parent()
ModularForms(n=3, k=36, ep=1) over Integer Ring
sage: e12 == constructed_e12
True
```

```python
sage: QF = QuasiModularForms(k=2)
sage: q = QF.get_q()
sage: qexp3 = 1 + O(q)
sage: QF(qexp3)
1 - 24*q - 72*q^2 - 96*q^3 - 168*q^4 + O(q^5)
sage: QF(qexp3) == QF.E2()
```

(continues on next page)
sage: QF = QuasiWeakModularForms(n=∞, k=2, ep=-1)
sage: el4 = QF.f_i() + QF.f_i()^3/QF.E4()
sage: prec = QF.required_laurent_prec(order_1=-1)
sage: qexp4 = el4.q_expansion(prec=prec)
sage: qexp4
2 - 7/(4*d)*q + 195/(256*d^2)*q^2 - 903/(4096*d^3)*q^3 + 41987/(1048576*d^4)*q^4
   - 181269/(33554432*d^5)*q^5 + O(q^6)
sage: QF.construct_quasi_form(qexp4, check=False) == el4
False
sage: QF.construct_quasi_form(qexp4, order_1=-1) == el4
True
sage: QF = QuasiCuspForms(n=8, k=22/3, ep=-1)
sage: el = QF(QF.f_inf()*QF.E2())
sage: qexp = el.q_expansion_fixed_d(d_num_prec=1000)
sage: qexp.parent()
Power Series Ring in q over Real Field with 1000 bits of precision
sage: QF.construct_quasi_form(qexp, rationalize=True) == el
True

construction()

Return a functor that constructs self (used by the coercion machinery).

EXAMPLES:

sage: from sage.modular.modform_hecketriangle.space import QuasiModularForms
sage: QuasiModularForms(n=4, k=2, ep=1, base_ring=CC).construction()
(QuasiModularFormsFunctor(n=4, k=2, ep=1), BaseFacade(Complex Field with 53 bits of precision))

sage: from sage.modular.modform_hecketriangle.space import ModularForms
sage: MF=ModularForms(k=12)
sage: MF.subspace([MF.gen(1)]).construction()
(FormsSubSpaceFunctor with 1 generator for the ModularFormsFunctor(n=3, k=12, ep=1), BaseFacade(Integer Ring))

contains_coeff_ring()

Return whether self contains its coefficient ring.

EXAMPLES:

sage: from sage.modular.modform_hecketriangle.space import QuasiModularForms
sage: QuasiModularForms(k=0, ep=-1, n=8).contains_coeff_ring()
True
sage: QuasiModularForms(k=0, ep=-1, n=8).contains_coeff_ring()
False

coordinate_vector(v)

This method should be overloaded by subclasses.

Return the coordinate vector of the element v with respect to self.gens().

NOTE:
Elements use this method (from their parent) to calculate their coordinates.

**INPUT:**

- v – An element of self.

**EXAMPLES:**

```python
sage: from sage.modular.modform_hecketriangle.space import ModularForms
sage: MF = ModularForms(n=4, k=24, ep=-1)
```

```python
sage: MF.coordinate_vector(MF.gen(0)).parent()  # defined in space.py
Vector space of dimension 3 over Fraction Field of Univariate Polynomial Ring
˓→ in d over Integer Ring
```

```python
sage: MF.coordinate_vector(MF.gen(0))  # defined in space.py
(1, 0, 0)
```

```python
sage: subspace = MF.subspace([MF.gen(0), MF.gen(2)])
```

```python
sage: subspace.coordinate_vector(subspace.gen(0)).parent()  # defined in
˓→subspace.py
Vector space of dimension 2 over Fraction Field of Univariate Polynomial Ring
˓→ in d over Integer Ring
```

```python
sage: subspace.coordinate_vector(subspace.gen(0))  # defined in
˓→subspace.py
(1, 0)
```

`degree()`

Return the degree of self.

**EXAMPLES:**

```python
sage: from sage.modular.modform_hecketriangle.space import ModularForms
sage: MF = ModularForms(n=4, k=24, ep=-1)
```

```python
sage: MF.degree()
3
```

```python
sage: MF.subspace([MF.gen(0), MF.gen(2)]).degree()  # defined in subspace.py
3
```

`dimension()`

Return the dimension of self.

**Note:** This method should be overloaded by subclasses.

**EXAMPLES:**

```python
sage: from sage.modular.modform_hecketriangle.space import ModularForms
```

```python
sage: from sage.modular.modform_hecketriangle.space import QuasiMeromorphicModularForms
```

```python
sage: QuasiMeromorphicModularForms(k=2, ep=-1).dimension()
+Infinity
```

`element_from_ambient_coordinates(vec)`

If self has an associated free module, then return the element of self corresponding to the given vec. Otherwise raise an exception.

**INPUT:**

- vec – An element of self.module() or self.ambient_module().
OUTPUT:
An element of self corresponding to vec.

EXAMPLES:

```python
sage: from sage.modular.modform_hecketriangle.space import ModularForms
sage: MF = ModularForms(k=24)
sage: MF.dimension()
3
sage: el = MF.element_from_ambient_coordinates([1,1,1])
sage: el == MF.element_from_coordinates([1,1,1])
True
sage: el.parent() == MF
True

sage: subspace = MF.subspace([MF.gen(0), MF.gen(1)])
```

**element_from_coordinates(vec)**

If self has an associated free module, then return the element of self corresponding to the given coordinate vector vec. Otherwise raise an exception.

INPUT:

- vec – A coordinate vector with respect to self.gens().

OUTPUT:

An element of self corresponding to the coordinate vector vec.

EXAMPLES:

```python
sage: from sage.modular.modform_hecketriangle.space import ModularForms
sage: MF = ModularForms(k=24)
sage: MF.dimension()
3
sage: el = MF.element_from_coordinates([1,1,1])
sage: el
1 + q + q^2 + 52611612*q^3 + 39019413208*q^4 + O(q^5)
sage: el == MF.gen(0) + MF.gen(1) + MF.gen(2)
True
sage: el.parent() == MF
True

sage: subspace = MF.subspace([MF.gen(0), MF.gen(1)])
```

```sql
2.3. Modular forms for Hecke triangle groups
```
Return the multiplier of (elements of) self.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.space import QuasiModularForms
sage: QuasiModularForms(n=16, k=16/7, ep=-1).ep()
-1
```

```
faber_pol(m, order_1=0, fix_d=False, d_num_prec=None)
```

If \( n=\infty \) a non-trivial order of \(-1\) can be specified through the parameter \( \text{order}_1 \) (default: 0). Otherwise it is ignored. Return the \( m \)'th Faber polynomial of \( \text{self} \) with a different normalization based on \( j_{\text{inv}} \) instead of \( J_{\text{inv}} \).

Namely a polynomial \( p(q) \) such that \( p(j_{\text{inv}})F_{\text{simple}}() \) has a Fourier expansion of the form \( q^m + O(q^{\text{order}_\infty + 1}) \), where \( \text{order}_\infty = \text{self}_\l1 - \text{order}_1 \) and \( p(q) \) is a monic polynomial of degree \( \text{order}_\infty - m \).

If \( n=\infty \) a non-trivial order of \(-1\) can be specified through the parameter \( \text{order}_1 \) (default: 0). Otherwise it is ignored.

The relation to \( \text{Faber}_\text{pol} \) is: \( \text{faber}_\text{pol}(q) = \text{Faber}_\text{pol}(d^aq) \).

INPUT:

- \( m \) – An integer \( m \leq \text{self}_\l1 - \text{order}_1 \).
- \( \text{order}_1 \) – The order at \(-1\) of \( F_{\text{simple}} \) (default: 0).
  This parameter is ignored if \( n \neq \infty \).
- \( \text{fix}_d \) – If False (default) a formal parameter is used for \( d \).
  If True then the numerical value of \( d \) is used (resp. an exact value if the group is arithmetic).
  Otherwise the given value is used for \( d \).
- \( \text{d_num_prec} \) – The precision to be used if a numerical value for \( d \) is substituted.
  Default: None in which case the default numerical precision of \( \text{self}.\text{parent}() \) is used.

OUTPUT:

The corresponding Faber polynomial \( p(q) \).

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.space import WeakModularForms
sage: MF = WeakModularForms(n=5, k=62/3, ep=-1)
sage: MF.weight_parameters()
(2, 3)
sage: MF.faber_pol(2)
1
sage: MF.faber_pol(1)
q - 19/(100*d)
sage: MF.faber_pol(0)
q^2 - 117/(200*d)*q + 9113/(320000*d^2)
sage: MF.faber_pol(-2)
q^4 - 11/(8*d)*q^3 + 41013/(800000*d^2)*q^2 - 2251291/(48000000*d^3)*q +
   1974089431/(4915200000000*d^4)
sage: (MF.faber_pol(2)(MF.j_inv())*MF.F_simple()).q_expansion(prec=MF._l1+2)
q^2 - 41/(200*d)*q^3 + O(q^4)
```

(continues on next page)
sage: (MF.faber_pol(1)(MF.j_inv())*MF.F_simple()).q_expansion(prec=MF._l1+1)
q + O(q^3)
sage: (MF.faber_pol(0)(MF.j_inv())*MF.F_simple()).q_expansion(prec=MF._l1+1)
1 + O(q^3)
sage: (MF.faber_pol(-2)(MF.j_inv())*MF.F_simple()).q_expansion(prec=MF._l1+1)
q^-2 + O(q^3)
sage: MF = WeakModularForms(n=4, k=-2, ep=1)
sage: MF.weight_parameters()
(-1, 3)
sage: MF.faber_pol(-1)
1
sage: MF.faber_pol(-2, fix_d=True)
q - 184
sage: MF.faber_pol(-3, fix_d=True)
q^2 - 288*q + 14364
sage: (MF.faber_pol(-1, fix_d=True)(MF.j_inv())*MF.F_simple()).q_expansion(prec=MF._l1+2, fix_d=True)
q^-1 + 80 + O(q)
sage: (MF.faber_pol(-2, fix_d=True)(MF.j_inv())*MF.F_simple()).q_expansion(prec=MF._l1+2, fix_d=True)
q^-2 + 400 + O(q)
sage: (MF.faber_pol(-3)(MF.j_inv())*MF.F_simple()).q_expansion(prec=MF._l1+2, fix_d=True)
q^-3 + 2240 + O(q)
sage: MF = WeakModularForms(n=infinity, k=14, ep=-1)
sage: MF.faber_pol(3)
1
sage: MF.faber_pol(2)
q + 3/(8*d)
sage: MF.faber_pol(1)
q^2 + 75/(1024*d^2)
sage: MF.faber_pol(0)
q^3 - 3/(8*d)*q^2 + 3/(512*d^2)*q + 41/(4096*d^3)
sage: (MF.faber_pol(-1)(MF.j_inv())*MF.F_simple()).q_expansion(prec=MF._l1 + 1)
q^-1 - 9075/(8388608*d^4)*q^3 + O(q^4)
sage: MF.faber_pol(3, order_1=-1)
q + 3/(4*d)
sage: MF.faber_pol(1, order_1=2)
1
sage: MF.faber_pol(0, order_1=2)
q - 3/(8*d)
sage: MF.faber_pol(-1, order_1=2)
q^2 - 3/(4*d)*q + 81/(1024*d^2)
sage: (MF.faber_pol(-1, order_1=2)(MF.j_inv())*MF.F_simple(order_1=2)).q_expansion(prec=MF._l1 + 1)
q^-1 - 9075/(8388608*d^4)*q^3 + 0(q^4)
gen($k=0$)

Return the $k$'th basis element of self if possible (default: $k=0$).

EXAMPLES:

\begin{verbatim}
sage: from sage.modular.modform_hecketriangle.space import ModularForms
dsage: ModularForms(k=12).gen(1).parent()
ModularForms(n=3, k=12, ep=1) over Integer Ring
dsage: ModularForms(k=12).gen(1)
q - 24*q^2 + 252*q^3 - 1472*q^4 + O(q^5)
\end{verbatim}

gens()

This method should be overloaded by subclasses.

Return a basis of self.

Note that the coordinate vector of elements of self are with respect to this basis.

EXAMPLES:

\begin{verbatim}
sage: from sage.modular.modform_hecketriangle.space import ModularForms
dsage: ModularForms(k=12).gens()
# defined in space.py
[1 + 196560*q^2 + 16773120*q^3 + 398034000*q^4 + O(q^5),
 q - 24*q^2 + 252*q^3 - 1472*q^4 + O(q^5)]
\end{verbatim}

homogeneous_part($k, ep$)

Since self already is a homogeneous component return self unless the degree differs in which case a ValueError is raised.

EXAMPLES:

\begin{verbatim}
sage: from sage.modular.modform_hecketriangle.space import _˓→QuasiMeromorphicModularForms
dsage: MF = QuasiMeromorphicModularForms(n=6, k=4)
dsage: MF == MF.homogeneous_part(4,1)
True
dsage: MF.homogeneous_part(5,1)
Traceback (most recent call last):
  ... ValueError: QuasiMeromorphicModularForms(n=6, k=4, ep=1) over Integer Ring˓→already is homogeneous with degree (4, 1) != (5, 1)!
\end{verbatim}

is_ambient()

Return whether self is an ambient space.

EXAMPLES:

\begin{verbatim}
sage: from sage.modular.modform_hecketriangle.space import ModularForms
dsage: MF = ModularForms(k=12)
dsage: MF.is_ambient()
True
dsage: MF.subspace([MF.gen(0)]).is_ambient()
False
\end{verbatim}

module()

Return the module associated to self.
EXAMPLES:

```python
sage: from sage.modular.modform_hecketriangle.space import ModularForms
sage: MF = ModularForms(k=12)
sage: MF.module()
Vector space of dimension 2 over Fraction Field of Univariate Polynomial Ring in d over Integer Ring
sage: subspace = MF.subspace([MF.gen(0)])
sage: subspace.module()
Vector space of degree 2 and dimension 1 over Fraction Field of Univariate Polynomial Ring in d over Integer Ring
Basis matrix:
[1 0]
```

`one()`

Return the one element from the corresponding space of constant forms.

**Note:** The one element does not lie in `self` in general.

EXAMPLES:

```python
sage: from sage.modular.modform_hecketriangle.space import CuspForms
sage: MF = CuspForms(k=12)
sage: MF.Delta()^0 == MF.one()
True
sage: (MF.Delta()^0).parent()
ModularForms(n=3, k=0, ep=1) over Integer Ring
```

`q_basis(m=None, min_exp=0, order_l=0)`

Try to return a (basis) element of `self` with a Laurent series of the form $q^m + O(q^N)$, where $N=\text{self. required_laurent_prec(min_exp)}$.

If `m=None` the whole basis (with varying $m$’s) is returned if it exists.

**INPUT:**

- `m` – An integer, indicating the desired initial Laurent exponent of the element.
  - If `m=None` (default) then the whole basis is returned.
- `min_exp` – An integer, indicating the minimal Laurent exponent (for each quasi part) of the subspace of `self` which should be considered (default: 0).
- `order_l` – A lower bound for the order at $-1$ of all quasi parts of the subspace (default: 0). If $n!=\text{infinity}$ this parameter is ignored.

**OUTPUT:**

The corresponding basis (if `m=None`) resp. the corresponding basis vector (if $m!=None$). If the basis resp. element doesn’t exist an exception is raised.

EXAMPLES:

```python
sage: from sage.modular.modform_hecketriangle.space import QuasiWeakModularForms, ModularForms, QuasiModularForms
sage: QF = QuasiWeakModularForms(n=8, k=10/3, ep=-1)
sage: QF.default_prec(QF.required_laurent_prec(min_exp=-1))
```

(continues on next page)
sage: q_basis = QF.q_basis(min_exp=-1)
sage: q_basis
[q^-1 + O(q^5), 1 + O(q^5), q + O(q^5), q^2 + O(q^5), q^3 + O(q^5), q^4 + O(q^5)]
sage: QF.q_basis(m=-1, min_exp=-1)
q^-1 + O(q^5)
sage: MF = ModularForms(k=36)
sage: MF.q_basis() == MF.gens()
True
sage: QF = QuasiModularForms(k=6)
sage: QF.required_laurent_prec()
3
sage: QF.q_basis()
[1 - 20160*q^3 - 158760*q^4 + O(q^5), q^-1 - 60*q^3 - 248*q^4 + O(q^5), q^2 + 8*q^3 + 30*q^4 + O(q^5)]
sage: QF = QuasiWeakModularForms(n=infinity, k=-2, ep=-1)
sage: QF.q_basis(order_1=-1)
[1 - 168*q^2 + 2304*q^3 - 19320*q^4 + O(q^5), q - 18*q^2 + 180*q^3 - 1316*q^4 + O(q^5)]

quasi_part_dimension(r=None, min_exp=0, max_exp=+Infinity, order_1=0)

Return the dimension of the subspace of self generated by self.quasi_part_gens(r, min_exp, max_exp, order_1).

See quasi_part_gens() for more details.

EXAMPLES:

sage: from sage.modular.modform_hecketriangle.space import QuasiModularForms, QuasiCuspForms, QuasiWeakModularForms
sage: MF = QuasiModularForms(n=5, k=6, ep=-1)
sage: [v.as_ring_element() for v in MF.gens()]
[f_rho^2*f_i, f_rho^3*E2, E2^3]
sage: MF.dimension()
3
sage: MF.quasi_part_dimension(r=0)
1
sage: MF.quasi_part_dimension(r=1)
1
sage: MF.quasi_part_dimension(r=2)
0
sage: MF.quasi_part_dimension(r=3)
1
sage: MF = QuasiCuspForms(n=5, k=18, ep=-1)
sage: MF.dimension()
8
sage: MF.quasi_part_dimension(r=0)
2
sage: MF.quasi_part_dimension(r=1)
sage: MF = QuasiCuspForms(n=infinity, k=18, ep=-1)
sage: MF.quasi_part_dimension(r=1, min_exp=-2)
3
sage: MF = QuasiWeakModularForms(n=infinity, k=4, ep=1)
sage: MF.quasi_part_dimension(min_exp=2, order_1=-2)
4
sage: [v.order_at(-1) for v in MF.quasi_part_gens(r=0, min_exp=2, order_1=-2)]
[-2, -2]

quasi_part_gens \( (r=\text{None}, \text{min}\_\text{exp}=0, \text{max}\_\text{exp}=+\text{Infinity}, \text{order}\_1=0) \)

Return a basis in self of the subspace of (quasi) weakly holomorphic forms which satisfy the specified properties on the quasi parts and the initial Fourier coefficient.

**INPUT:**

- **r** – An integer or None (default), indicating
  the desired power of \( E2 \) If \( r=\text{None} \) then all possible powers \( (r) \) are choosen.

- **min_exp** – An integer giving a lower bound for the
  first non-trivial Fourier coefficient of the generators (default: 0).

- **max_exp** – An integer or infinity (default) giving
  an upper bound for the first non-trivial Fourier coefficient of the generators. If
  \( \text{max}\_\text{exp}==\text{infinity} \) then no upper bound is assumed.

- **order_1** – A lower bound for the order at \(-1\) of all
  quasi parts of the basis elements (default: 0). If \( n!=\text{infinity} \) this parameter is ignored.

**OUTPUT:**

A basis in self of the subspace of forms which are modular after dividing by \( E2^r \) and which have a Fourier expansion of the form \( q^m + \ldots + O(q^{m+1}) \) with \( \text{min}\_\text{exp} <= m <= \text{max}\_\text{exp} \) for each quasi part (and at least the specified order at \(-1\) in case \( n=\text{infinity} \)). Note that linear combinations of forms/quasi parts may have a higher order at infinity than \( \text{max}\_\text{exp} \).

**EXAMPLES:**

sage: from sage.modular.modform_hecketriangle.space import QuasiWeakModularForms
sage: QF = QuasiWeakModularForms(n=8, k=10/3, ep=-1)

(continues on next page)
sage: QF.default_prec(1)
sage: QF.quasi_part_gens(min_exp=-1)
[q^-1 + O(q), 1 + O(q), q^-1 - 9/(128*d) + O(q), 1 + O(q), q^-1 - 19/(64*d) + O(q), q^-1 + 1/(64*d) + O(q)]
sage: QF.quasi_part_gens(min_exp=-1, max_exp=-1)
[q^-1 - 9/(128*d) + O(q), q^-1 - 19/(64*d) + O(q), q^-1 + 1/(64*d) + O(q)]
sage: QF.quasi_part_gens(min_exp=-2, r=1)
[q^-2 - 9/(128*d)*q^-1 - 261/(131072*d^2) + O(q), q^-1 - 9/(128*d) + O(q), 1 + O(q)]

sage: from sage.modular.modform_hecketriangle.space import ModularForms
sage: MF = ModularForms(k=36)
sage: MF.quasi_part_gens(min_exp=2)
[q^2 + 194184*q^4 + O(q^5), q^3 - 72*q^4 + O(q^5)]

sage: from sage.modular.modform_hecketriangle.space import QuasiModularForms
sage: MF = QuasiModularForms(n=5, k=6, ep=-1)
sage: MF.default_prec(2)
sage: MF.dimension()
3
sage: MF.quasi_part_gens(r=0)
[1 - 37/(200*d)*q + O(q^2)]
sage: MF.quasi_part_gens(r=0)[0] == MF.E6()
True
sage: MF.quasi_part_gens(r=1)
[1 + 33/(200*d)*q + O(q^2)]
sage: MF.quasi_part_gens(r=1)[0] == MF.E2()*MF.E4()
True
sage: MF.quasi_part_gens(r=2)
[]
sage: MF.quasi_part_gens(r=3)
[1 - 27/(200*d)*q + O(q^2)]
sage: MF.quasi_part_gens(r=3)[0] == MF.E2()^3
True

sage: from sage.modular.modform_hecketriangle.space import QuasiCuspForms, CuspForms
sage: MF = QuasiCuspForms(n=5, k=18, ep=-1)
sage: MF.default_prec(4)
sage: MF.dimension()
8
sage: MF.quasi_part_gens(r=0)
[q - 34743/(640000*d^2)*q^3 + O(q^4), q^2 - 69/(200*d)*q^3 + O(q^4)]
sage: MF.quasi_part_gens(r=1)
[q - 9/(200*d)*q^2 + 37633/(640000*d^2)*q^3 + O(q^4), q^2 + 1/(200*d)*q^3 + O(q^4)]
sage: MF.quasi_part_gens(r=2)
[q - 1/(4*d)*q^2 - 24903/(640000*d^2)*q^3 + O(q^4)]
sage: MF.quasi_part_gens(r=3)
[q + 1/(10*d)*q^2 - 7263/(640000*d^2)*q^3 + O(q^4)]
sage: MF.quasi_part_gens(r=4)
[q - 11/(20*d)*q^2 + 53577/(640000*d^2)*q^3 + O(q^4)]
sage: MF.quasi_part_gens(r=5)
[q - 1/(5*d)*q^2 + 4017/(640000*d^2)*q^3 + O(q^4)]
sage: MF.quasi_part_gens(r=1)[0] == MF.E2() * CuspForms(n=5, k=16, ep=1).gen(0)
True
sage: MF.quasi_part_gens(r=1)[1] == MF.E2() * CuspForms(n=5, k=16, ep=1).gen(1)
True
sage: MF.quasi_part_gens(r=3)[0] == MF.E2()^3 * MF.Delta()
True
sage: MF = QuasiCuspForms(n=infinity, k=18, ep=-1)
sage: MF.quasi_part_gens(r=1, min_exp=-2) == MF.quasi_part_gens(r=1, min_exp=1)
True
sage: MF.quasi_part_gens(r=1)
[q - 8*q^2 - 8*q^3 + 5952*q^4 + O(q^5),
  q^2 - 8*q^3 + 208*q^4 + O(q^5),
  q^3 - 16*q^4 + O(q^5)]
sage: MF = QuasiWeakModularForms(n=infinity, k=4, ep=1)
sage: MF.quasi_part_gens(r=2, min_exp=2, order_1=-2)[0] == MF.E2()^2 * MF.E4()^(-2) * MF.f_inf()^2
True
sage: [v.order_at(-1) for v in MF.quasi_part_gens(r=0, min_exp=2, order_1=-2)]
[-2, -2]

**rank()**

Return the rank of self.

**EXAMPLES:**

```python
sage: from sage.modular.modform hecketriangle.space import ModularForms
sage: MF = ModularForms(n=4, k=24, ep=-1)
sage: MF.rank()
3
sage: MF.subspace([MF.gen(0), MF.gen(2)]).rank()
2
```

**rationalize_series**(laurent_series, coef_bound=1e-10, denom_factor=1)

Try to return a Laurent series with coefficients in self.coef_ring() that matches the given Laurent series.

We give our best but there is absolutely no guarantee that it will work!

**INPUT:**

- **laurent_series** – A Laurent series. If the Laurent coefficients already coerce into self.coef_ring() with a formal parameter then the Laurent series is returned as is.

  Otherwise it is assumed that the series is normalized in the sense that the first non-trivial coefficient is a power of $d$ (e.g. 1).

- **coef_bound** – Either None resp. 0 or a positive real number

**2.3. Modular forms for Hecke triangle groups**
(default: \(1e-10\)). If specified `coeff_bound` gives a lower bound for the size of the initial Laurent coefficients. If a coefficient is smaller it is assumed to be zero.

For calculations with very small coefficients (less than \(1e-10\)) `coeff_bound` should be set to something even smaller or just 0.

Non-exact calculations often produce non-zero coefficients which are supposed to be zero. In those cases this parameter helps a lot.

- **denom_factor** – An integer (default: 1) whose factor might occur in the denominator of the given Laurent coefficients (in addition to naturally occurring factors).

**OUTPUT:**

A Laurent series over `self.coeff_ring()` corresponding to the given Laurent series.

**EXAMPLES:**

```python
sage: from sage.modular.modform hecketriangle.space import WeakModularForms
sage: WF = WeakModularForms(n=14)
sage: qexp = WF.J_inv().q_expansion_fixed_d(d_num_prec=1000)
sage: qexp.parent()
Laurent Series Ring in q over Real Field with 1000 bits of precision
sage: qexp_int = WF.rationalize_series(qexp)
sage: qexp_int.add_bigoh(3)
d*q^-1 + 37/98 + 2587/(38416*d)*q + 899/(117649*d^2)*q^2 + O(q^3)
sage: qexp_int == WF.J_inv().q_expansion()
True
sage: WF.rationalize_series(qexp_int) == qexp_int
True
sage: WF(qexp_int) == WF.J_inv()
True
sage: WF.rationalize_series(qexp.parent()(1))
1
sage: WF.rationalize_series(qexp_int.parent()(1)).parent()
Laurent Series Ring in q over Fraction Field of Univariate Polynomial Ring in d over Integer Ring
sage: MF = ModularForms(n=infinity, k=4)
sage: qexp = MF.E4().q_expansion_fixed_d()
sage: qexp.parent()
Power Series Ring in q over Rational Field
sage: qexp_int = MF.rationalize_series(qexp)
sage: qexp_int.parent()
Power Series Ring in q over Fraction Field of Univariate Polynomial Ring in d over Integer Ring
sage: qexp_int == MF.E4().q_expansion()
True
sage: MF(rationalize_series(qexp_int)) == qexp_int
True
sage: MF(qexp_int) == MF.E4()
True
sage: QF = QuasiCuspForms(n=8, k=22/3, ep=-1)
```

(continues on next page)
sage: el = QF(QF.f_inf()*QF.E2())
sage: qexp = el.q_expansion_fixed_d(d_num_prec=1000)
sage: qexp.parent()
Power Series Ring in q over Real Field with 1000 bits of precision
sage: qexp_int = QF.rationalize_series(qexp)
sage: qexp_int.parent()
Power Series Ring in q over Fraction Field of Univariate Polynomial Ring in d
˓→ over Integer Ring
sage: qexp_int == el.q_expansion()
True
sage: QF.rationalize_series(qexp_int) == qexp_int
True
sage: QF(qexp_int) == el
True

required_laurent_prec(min_exp=0, order_1=0)

Return an upper bound for the required precision for Laurent series to uniquely determine a corresponding
(quasi) form in self with the given lower bound min_exp for the order at infinity (for each quasi part).

Note: For n=\infty only the holomorphic case (min_exp \geq 0) is supported (in particular a non-
negative order at -1 is assumed).

INPUT:

• min_exp – An integer (default: 0), namely the lower bound for the
  order at infinity resp. the exponent of the Laurent series.

• order_1 – A lower bound for the order at -1 for all quasi parts
  (default: 0). If n!=\infty this parameter is ignored.

OUTPUT:

An integer, namely an upper bound for the number of required Laurent coefficients. The bound should be
precise or at least pretty sharp.

EXAMPLES:

sage: from sage.modular.modform_hecketriangle.space import_
˓→QuasiWeakModularForms, ModularForms, QuasiModularForms
sage: QF = QuasiWeakModularForms(n=8, k=10/3, ep=-1)
sage: QF.required_laurent_prec(min_exp=-1)
5
sage: MF = ModularForms(k=36)
sage: MF.required_laurent_prec(min_exp=2)
4
sage: QuasiModularForms(k=2).required_laurent_prec()
1
sage: QuasiWeakModularForms(n=\infty, k=2, ep=-1).required_laurent_prec(order_˓→1=-1)
6

2.3. Modular forms for Hecke triangle groups
subspace(basis)
    Return the subspace of self generated by basis.

EXAMPLES:

    sage: from sage.modular.modform_hecketriangle.space import ModularForms
    sage: MF = ModularForms(k=24)
    sage: MF.dimension()
    3
    sage: subspace = MF.subspace([MF.gen(0), MF.gen(1)])
    sage: subspace
    Subspace of dimension 2 of ModularForms(n=3, k=24, ep=1) over Integer Ring

weight()
    Return the weight of (elements of) self.

EXAMPLES:

    sage: from sage.modular.modform_hecketriangle.space import QuasiModularForms
    sage: QuasiModularForms(n=16, k=16/7, ep=-1).weight()
    16/7

weight_parameters()
    Check whether self has a valid weight and multiplier.

If not then an exception is raised. Otherwise the two weight parameters
    corresponding to the weight and multiplier of self are returned.

The weight parameters are e.g. used to calculate dimensions or
    precisions of Fourier expansion.

EXAMPLES:

    sage: from sage.modular.modform_hecketriangle.space import MeromorphicModularForms
    sage: MF = MeromorphicModularForms(n=18, k=-7, ep=-1)
    sage: MF.weight_parameters()  
    (-3, 17)
    sage: (MF._l1, MF._l2) == MF.weight_parameters()  
    True
    sage: (k, ep) = (MF.weight(), MF.ep())
    sage: n = MF.hecke_n()
    sage: k == 4^(n^2*MF._l1 + MF._l2)/(n-2) + (1-ep)^n/(n-2)
    True
    sage: from sage.modular.modform_hecketriangle.space import ModularForms
    sage: MF = ModularForms(n=5, k=12, ep=1)
    sage: MF.weight_parameters()  
    (1, 4)
    sage: (MF._l1, MF._l2) == MF.weight_parameters()  
    True
    sage: (k, ep) = (MF.weight(), MF.ep())
    sage: n = MF.hecke_n()
    sage: k == 4^(n^2*MF._l1 + MF._l2)/(n-2) + (1-ep)^n/(n-2)
    True
    sage: MF.dimension() == MF._l1 + 1
    True
sage: MF = ModularForms(n=infinity, k=8, ep=1)
sage: MF.weight_parameters()
(2, 0)
sage: MF.dimension() == MF._l1 + 1
True

2.4 Elements of Hecke modular forms spaces

AUTHORS:
• Jonas Jermann (2013): initial version

class sage.modular.modform_hecketriangle.element.FormsElement(parent, rat)

Bases: FormsRingElement

(Hecke) modular forms.

ambient_coordinate_vector()

Return the coordinate vector of self with respect to self.parent().ambient_space().gens(). The returned coordinate vector is an element of self.parent().module().

Note: This uses the corresponding function of the parent. If the parent has not defined a coordinate vector function or an ambient module for coordinate vectors then an exception is raised by the parent (default implementation).

EXAMPLES:

sage: from sage.modular.modform_hecketriangle.space import ModularForms
sage: MF = ModularForms(n=4, k=24, ep=-1)
sage: MF.gen(0).ambient_coordinate_vector().parent()
Vector space of dimension 3 over Fraction Field of Univariate Polynomial Ring...
→ in d over Integer Ring
sage: MF.gen(0).ambient_coordinate_vector()
(1, 0, 0)
sage: subspace = MF.subspace([MF.gen(0), MF.gen(2)])
sage: subspace.gen(0).ambient_coordinate_vector().parent()
Vector space of degree 3 and dimension 2 over Fraction Field of Univariate...
→ Polynomial Ring in d over Integer Ring
Basis matrix:
[1 0 0]
[0 0 1]
sage: subspace.gen(0).ambient_coordinate_vector()
(1, 0, 0)
sage: subspace.gen(0).ambient_coordinate_vector() == subspace.ambient_coordinate_vector(subspace.gen(0))
True

cordinate_vector()

Return the coordinate vector of self with respect to self.parent().gens().

2.4. Elements of Hecke modular forms spaces 171
Note: This uses the corresponding function of the parent. If the parent has not defined a coordinate vector function or a module for coordinate vectors then an exception is raised by the parent (default implementation).

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.space import ModularForms
sage: MF = ModularForms(n=4, k=24, ep=-1)
sage: MF.gen(0).coordinate_vector().parent()
Vector space of dimension 3 over Fraction Field of Univariate Polynomial Ring
˓→in d over Integer Ring
sage: MF.gen(0).coordinate_vector()
(1, 0, 0)
sage: subspace = MF.subspace([MF.gen(0), MF.gen(2)])
sage: subspace.gen(0).coordinate_vector().parent()
Vector space of dimension 2 over Fraction Field of Univariate Polynomial Ring
˓→in d over Integer Ring
sage: subspace.gen(0).coordinate_vector()
(1, 0)
sage: subspace.gen(0).coordinate_vector() == subspace.coordinate_˓→vector(subspace.gen(0))
True
```

lseries(num_prec=None, max_imaginary_part=0, max_asymp_coeffs=40)

Return the L-series of self if self is modular and holomorphic.

This relies on the (pari) based function Dokchitser.

INPUT:

- num_prec – An integer denoting the to-be-used numerical precision.
  - If integer num_prec=None (default) the default numerical precision of the parent of self is used.

- max_imaginary_part – A real number (default: 0), indicating up to which imaginary part the L-series is going to be studied.

- max_asymp_coeffs – An integer (default: 40).

OUTPUT:

An interface to Tim Dokchitser’s program for computing L-series, namely the series given by the Fourier coefficients of self.

EXAMPLES:

```
sage: from sage.modular.modform.eis_series import eisenstein_series_lseries
case: from sage.modular.modform_hecketriangle.space import ModularForms
case: f = ModularForms(n=3, k=4).E4()/240
case: L = f.lseries()
case: L
L-series associated to the modular form 1/240 + q + 9*q^2 + 28*q^3 + 73*q^4 +˓→O(q^5)
case: L.conductor
1
case: L(i).prec()
53
```
```python
sage: L.check_functional_equation() < 2^(-50)
True
sage: L(1)
-0.0304484570583...

sage: abs(L(1) - eisenstein_series_lseries(4)(1)) < 2^(-53)
True
sage: L.derivative(1, 1)
-0.0504570844798...

sage: L(10)
1.00935215649...
```

(continues from previous page)
2.5 Elements of graded rings of modular forms for Hecke triangle groups

AUTHORS:

• Jonas Jermann (2013): initial version

class sage.modular.modform hecketriangle.graded_ring_element.FormsRingElement(parent, rat)

Bases: CommutativeAlgebraElement, UniqueRepresentation

Element of a FormsRing.

AT = Analytic Type

AnalyticType

alias of AnalyticType

analytic_type()

Return the analytic type of self.

EXAMPLES:

sage: from sage.modular.modform hecketriangle.graded_ring_element import _

QuasiMeromorphicModularFormsRing
sage: from sage.modular.modform hecketriangle.space import _

QuasiMeromorphicModularForms
sage: x,y,z,d = var("x,y,z,d")

sage: QuasiMeromorphicModularFormsRing(n=5)(x/z+d).analytic_type()
quasi meromorphic modular
sage: QuasiMeromorphicModularFormsRing(n=5)((y^3-z^5)/(x^5-y^2)+y-d).analytic_
quasi weakly holomorphic modular
sage: QuasiMeromorphicModularFormsRing(n=5)(x^2+y-d).analytic_type()
modular
sage: QuasiMeromorphicModularForms(n=18).J_inv().analytic_type()
weakly holomorphic modular
sage: QuasiMeromorphicModularForms(n=18).f_inf().analytic_type()
cuspidal
sage: QuasiMeromorphicModularForms(n=infinity).f_inf().analytic_type()
modular

as_ring_element()
Coerce self into the graded ring of its parent.

EXAMPLES:

sage: from sage.modular.modform_hecketriangle.space import CuspForms
sage: Delta = CuspForms(k=12).Delta()
sage: Delta.parent()
CuspForms(n=3, k=12, ep=1) over Integer Ring
sage: Delta.as_ring_element()
f_rho^3*d - f_i^2*d
sage: Delta.as_ring_element().parent()
CuspFormsRing(n=3) over Integer Ring
sage: CuspForms(n=infinity, k=12).Delta().as_ring_element()
-E4^2*f_i^2*d + E4^3*d

base_ring()
Return base ring of self.parent().

EXAMPLES:

sage: from sage.modular.modform_hecketriangle.space import ModularForms
sage: ModularForms(n=12, k=4, base_ring=CC).E4().base_ring()
Complex Field with 53 bits of precision

coeff_ring()
Return coefficient ring of self.

EXAMPLES:

sage: from sage.modular.modform_hecketriangle.graded_ring import ModularFormsRing
sage: ModularFormsRing().E6().coeff_ring()
Fraction Field of Univariate Polynomial Ring in d over Integer Ring

degree()
Return the degree of self in the graded ring. If self is not homogeneous, then (None, None) is returned.

EXAMPLES:

sage: from sage.modular.modform_hecketriangle.graded_ring import ModularFormsRing
sage: ModularFormsRing().E6().degree()
(0, None)

(continues on next page)
sage: x,y,z,d = var("x,y,z,d")

sage: QuasiModularFormsRing()(x+y).degree() == (None, None)
True

sage: ModularForms(n=18).f_i().degree()
(9/4, -1)

sage: ModularForms(n=infinity).f_rho().degree()
(0, 1)

denominator()

Return the denominator of self. I.e. the (properly reduced) new form corresponding to the numerator of self.rat().

Note that the parent of self might (probably will) change.

EXAMPLES:

sage: from sage.modular.modform_hecketriangle.graded_ring import QuasiMeromorphicModularFormsRing
sage: from sage.modular.modform_hecketriangle.space import QuasiMeromorphicModularForms
sage: x,y,z,d = var("x,y,z,d")

sage: QuasiMeromorphicModularFormsRing(n=5).Delta().full_reduce().denominator()
1 + O(q^5)

sage: QuasiMeromorphicModularFormsRing(n=5)((y^3-z^5)/(x^5-y^2)+y-d).denominator()
f_rho^5 - f_i^2

sage: QuasiMeromorphicModularFormsRing(n=5)((y^3-z^5)/(x^5-y^2)+y-d).denominator().parent()
QuasiModularFormsRing(n=5) over Integer Ring

sage: QuasiMeromorphicModularForms(n=5, k=-2, ep=-1)(x/y).denominator()
1 - 13/(40*d)*q - 351/(64000*d^2)*q^2 - 13819/(76800000*d^3)*q^3 - 1163669/(491520000000*d^4)*q^4 + O(q^5)

sage: QuasiMeromorphicModularForms(n=5, k=10/3, ep=-1) over Integer Ring

sage: (QuasiMeromorphicModularForms(n=infinity, k=-6, ep=-1)(y/(x*(x-y^2))))
-64*q^6 + 768*q^4 - 8192*q^3 + 4096*q^2 + 0(q^5)

sage: (QuasiMeromorphicModularForms(n=infinity, k=-6, ep=-1)(y/(x*(x-y^2))))


derivative()

Return the derivative \(d/dq = \lambda/(2\pi i) \cdot d/\tau\) of self.

Note that the parent might (probably will) change. In particular its analytic type will be extended to contain "quasi".

If parent.has_reduce_hom() == True then the result is reduced to be an element of the corresponding forms space if possible.

In particular this is the case if self is a (homogeneous) element of a forms space.

EXAMPLES:
```
sage: from sage.modular.modform_hecketriangle.graded_ring import...
˓→QuasiMeromorphicModularFormsRing
sage: MR = QuasiMeromorphicModularFormsRing(n=7, red_hom=True)
sage: n = MR.hecke_n()
sage: E2 = MR.E2().full_reduce()
sage: E6 = MR.E6().full_reduce()
sage: f_rho = MR.f_rho().full_reduce()
sage: f_i = MR.f_i().full_reduce()
sage: f_inf = MR.f_inf().full_reduce()

sage: derivative(f_rho) == 1/n * (f_rho*E2 - f_i)
True
sage: derivative(f_i) == 1/2 * (f_i*E2 - f_rho**(n-1))
True
sage: derivative(f_inf) == f_inf * E2
True
sage: derivative(f_inf).parent()
QuasiCuspForms(n=7, k=38/5, ep=-1) over Integer Ring
sage: derivative(E2) == (n-2)/(4*n) * (E2**2 - f_rho**(n-2))
True
sage: derivative(E2).parent()
QuasiModularForms(n=+Infinity, k=6, ep=-1) over Integer Ring

sage: MR = QuasiMeromorphicModularFormsRing(n=+Infinity, red_hom=True)
sage: E2 = MR.E2().full_reduce()
sage: E4 = MR.E4().full_reduce()
sage: E6 = MR.E6().full_reduce()
sage: f_i = MR.f_i().full_reduce()
sage: f_inf = MR.f_inf().full_reduce()

sage: derivative(E4) == E4 * (E2 - f_i)
True
sage: derivative(f_i) == 1/2 * (f_i*E2 - E4)
True
sage: derivative(f_inf) == f_inf * E2
True
sage: derivative(f_inf).parent()
QuasiModularForms(n=+Infinity, k=6, ep=-1) over Integer Ring
sage: derivative(E2) == 1/4 * (E2**2 - E4)
True
sage: derivative(E2).parent()
QuasiModularForms(n=+Infinity, k=4, ep=1) over Integer Ring
```

**diff_op** *(op, new_parent=None)*

Return the differential operator op applied to self. If parent.has_reduce_hom() == True then the result is reduced to be an element of the corresponding forms space if possible.

**INPUT:**

- op – An element of self.parent().diff_alg().
  I.e. an element of the algebra over QQ of differential operators generated by X, Y, Z, dX, dY, DZ, where e.g. X corresponds to the multiplication by x (resp. f_rho) and dX corresponds to d/dx.
  To expect a homogeneous result after applying the operator to a homogeneous element it should be homogeneous operator (with respect to the usual, special grading).

---

2.5. Elements of graded rings of modular forms for Hecke triangle groups 177
• new_parent – Try to convert the result to the specified
  new_parent. If new_parent == None (default) then the parent is extended to a “quasi mero-
  morphic” ring.

OUTPUT:

The new element.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.graded_ring import QuasiMeromorphicModularFormsRing
sage: MR = QuasiMeromorphicModularFormsRing(n=8, red_hom=True)
sage: (X,Y,Z,dX,dY,dZ) = MR.diff_alg().gens()
sage: n = MR.hecke_n()
sage: mul_op = 4/(n-2)*X*dX + 2*n/(n-2)*Y*dY + 2*Z*dZ
sage: der_op = MR._derivative_op()
sage: ser_op = MR._serre_derivative_op()
sage: der_op == ser_op + (n-2)/(4*n)*Z*mul_op
True
sage: Delta = MR.Delta().full_reduce()
sage: E2 = MR.E2().full_reduce()
sage: Delta.diff_op(mul_op) == 12*Delta
True
sage: Delta.diff_op(Z*mul_op, Delta.parent().extend_type("quasi", ring=True)) == 12*E2*Delta
True
sage: ran_op = X + Y*X*dY*dX + dZ + dX^2
sage: Delta.diff_op(ran_op)
f_rhoe^19*d + 3046*f_rhoe^16*d - f_rhoe^11*f_i^2*d - 20*f_rhoe^10*f_i^2*d - 90*f_rhoe^8*f_i^2*d
sage: E2.diff_op(ran_op)
f_rhoe*E2 + 1
```

(continues on next page)
QuasiMeromorphicModularForms(n=+Infinity, k=12, ep=1) over Integer Ring
sage: Delta.diff_op(mul_op, Delta.parent()).parent()
CuspForms(n=+Infinity, k=12, ep=1) over Integer Ring
sage: E2.diff_op(mul_op, E2.parent()) == 2^Ep
True
sage: Delta.diff_op(Z*mul_op, Delta.parent().extend_type("quasi", ring=True))
== 12*E2*Delta
True
sage: ran_op = X + Y*X*dY*dX + dZ + dX^2
sage: Delta.diff_op(ran_op)
-E4^3*f_i^2*d + E4^4*d - 4*E4^2*f_i^2*d - 2*f_i^2*d + 6*E4*d
sage: E2.diff_op(ran_op)
E4*E2 + 1

ep()

Return the multiplier of self.

EXAMPLES:

sage: from sage.modular.modform.hecketriangle.graded_ring import QuasiModularFormsRing
sage: from sage.modular.modform.hecketriangle.space import ModularForms
sage: x,y,z,d = var("x,y,z,d")
sage: QuasiModularFormsRing()(x+y).ep() is None
True
sage: ModularForms(n=18).f_i().ep()
-1
sage: ModularForms(n=Infinity).E2().ep()
-1

evaluate(tau, prec=None, num_prec=None, check=False)

Try to return self evaluated at a point tau in the upper half plane, where self is interpreted as a function in tau, where q=exp(2*pi*i*tau).

Note that this interpretation might not make sense (and fail) for certain (many) choices of (base_ring, tau.parent()).

It is possible to evaluate at points of HyperbolicPlane(). In this case the coordinates of the upper half plane model are used.

To obtain a precise and fast result the parameters prec and num_prec both have to be considered/balanced. A high prec value is usually quite costly.

INPUT:

- **tau** – infinity or an element of the upper half plane. E.g. with parent AA or CC.
- **prec** – An integer, namely the precision used for the Fourier expansion. If prec == None (default) then the default precision of self.parent() is used.
- **num_prec** – An integer, namely the minimal numerical precision used for tau and d. If num_prec == None (default) then the default numerical precision of self.parent() is used.
• check – If True then the order of tau is checked.
   Otherwise the order is only considered for tau = infinity, i, rho, -1/rho. Default: False.

OUTPUT:
The (numerical) evaluated function value.

ALGORITHM:
1. If the order of self at tau is known and nonzero: Return 0 resp. infinity.
2. Else if tau==infinity and the order is zero: Return the constant Fourier coefficient of self.
3. Else if self is homogeneous and modular:
   1. Because of the (modular) transformation property of self the evaluation at tau is given by the
      evaluation at w multiplied by aut_factor(A,w).
   2. The evaluation at w is calculated by evaluating the truncated Fourier expansion of self at q(w).
      Note that this is much faster and more precise than a direct evaluation at tau.
4. Else if self is exactly E2:
   1. The same procedure as before is applied (with the aut_factor from the corresponding modular
      space).
   2. Except that at the end a correction term for the quasimodular form E2 of the form 4*lambda/
      (2*pi*i)*n/(n-2) * c*(c*w + d) (resp. 4/(pi*i) * c*(c*w + d) for n=infinity) has
      to be added, where lambda = 2*cos(pi/n) (resp lambda = 2 for n=infinity) and c,d are
      the lower entries of the matrix A.
5. Else:
   1. Evaluate f_rho, f_i, E2 at tau using the above procedures. If n=infinity use E4 instead of
      f_rho.
   2. Substitute x=f_rho(tau), y=f_i(tau), z=E2(tau) and the numerical value of d for d in
      self.rat(). If n=infinity then substitute x=E4(tau) instead.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.hecke_triangle_groups import HeckeTriangleGroup
sage: from sage.modular.modform_hecketriangle.graded_ring import QuasiMeromorphicModularFormsRing
sage: MR = QuasiMeromorphicModularFormsRing(n=5, red_hom=True)
sage: f_rho = MR.f_rho().full_reduce()
sage: f_i = MR.f_i().full_reduce()
sage: f_inf = MR.f_inf().full_reduce()
sage: E2 = MR.E2().full_reduce()
sage: E4 = MR.E4().full_reduce()
sage: rho = MR.group().rho()
sage: f_rho(rho) 0
sage: f_rho(rho + 1e-100) # since rho == rho + 1e-100 0
sage: f_rho(rho + 1e-6) 2.525...e-10 - 3.884...e-6*I
```
sage: f_i(i)
0
sage: f_i(i + 1e-1000)  # rel tol 5e-2
-6.08402217494586e-14 - 4.10147008296517e-1000*I
sage: f_inf(infinity)
0
sage: i = I = QuadraticField(-1, 'I').gen()
sage: z = -1/(-1/(2*i+30)-1)
sage: z
2/965*I + 934/965
sage: E4(z)
32288.05588811... - 118329.8566016...*I
sage: E4(z, prec=30, num_prec=100)  # long time
32288.05587235113041311053... - 118329.856600349999751420381...*I
sage: E2(z)
409.3144737105... + 100.6926857489...*I
sage: E2(z, prec=30, num_prec=100)  # long time
409.314473710489761254584951... + 100.692685748952440684513866...*I
sage: (E2^2-E4)(z)
125111.26553831962200469... + 200759.803948009905410385699...*I
sage: (E2^2-E4)(infinity)
0
sage: (1/(E2^2-E4))(infinity)
+Infinity
sage: ((E2^2-E4)/f_inf)(infinity)
-3/(10*d)

sage: G = HeckeTriangleGroup(n=8)
sage: MR = QuasiMeromorphicModularFormsRing(group=G, red_hom=True)
sage: f_rho = MR.f_rho().full_reduce()
sage: f_i = MR.f_i().full_reduce()
sage: E2 = MR.E2().full_reduce()
sage: z = AlgebraicField()(1/10+13/10*I)
sage: A = G.V(4)
sage: S = G.S()
sage: T = G.T()
sage: A == (T^S)**3*T
True
sage: az = A.acton(z)
sage: az == (A[0,0]*z + A[0,1]) / (A[1,0]*z + A[1,1])
True

sage: f_rho(z)
1.03740476727... + 0.0131941034523...*I
sage: f_rho(az)
-2.29216470688... - 1.46235057536...*I
sage: k = f_rho.weight()
sage: aut_fact = f_rho.ep()^3 * (((T*S)**2*T).acton(z)/AlgebraicField()(i))**k * ((T*S)*T).acton(z)/AlgebraicField()(i)**k * (T.acton(z)/AlgebraicField()(i))**k
sage: abs(aut_fact - f_rho.parent().aut_factor(A, z)) < 1e-12
True
sage: aut_fact * f_rho(z)
-2.29216470688... - 1.46235057536...*I
sage: f_rho.parent().default_num_prec(1000)
sage: f_rho.parent().default_prec(300)
sage: (f_rho.q_expansion_fixed_d().polynomial())(exp((2*pi*i).n(1000)*z/G.lam()))
1.037404767219462149821251... + 0.013194103452368974597290332...*I
sage: (f_rho.q_expansion_fixed_d().polynomial())(exp((2*pi*i).n(1000)*az/G.lam()))
-2.2921647068881834598616367... - 1.4623505753697635207183406...*I
sage: f_i(z)
0.667489320423... - 0.118902824870...*I
sage: f_i(az)
14.5845388476... - 28.4604652892...*I
sage: k = f_i.weight()
sage: aut_fact = f_i.ep()^3 * (((T*S)**2*T).acton(z)/AlgebraicField()(i))**k * ((T*S)*T).acton(z)/AlgebraicField()(i)**k * (T.acton(z)/AlgebraicField()(i))**k
sage: abs(aut_fact - f_i.parent().aut_factor(A, z)) < 1e-12
True
sage: aut_fact * f_i(z)
14.5845388476... - 28.4604652892...*I
sage: f_i.parent().default_num_prec(1000)
sage: f_i.parent().default_prec(300)
sage: (f_i.q_expansion_fixed_d().polynomial())(exp((2*pi*i).n(1000)*z/G.lam()))
0.66748932042300250077433252... - 0.11890282487028677063054267...*I
sage: (f_i.q_expansion_fixed_d().polynomial())(exp((2*pi*i).n(1000)*az/G.lam()))
14.584538847698600875918891... - 28.460465289220303834894855...*I
sage: f = f_rho*E2
sage: f(z)
0.966024386418... - 0.0138894699429...*I
sage: f(az)
-15.9978074989... - 29.2775758341...*I
sage: k = f.weight()
sage: aut_fact = f.ep()^3 * (((T*S)**2*T).acton(z)/AlgebraicField()(i))**k * ((T*S)*T).acton(z)/AlgebraicField()(i)**k * (T.acton(z)/AlgebraicField()(i))**k
sage: abs(aut_fact - f.parent().aut_factor(A, z)) < 1e-12
True
sage: k2 = f_rho.weight()
sage: aut_fact2 = f_rho.ep() * (((T*S)**2*T).acton(z)/AlgebraicField()(i))**k2
→* (((T*S)*T).acton(z)/AlgebraicField()(i))**k2 * (T.acton(z)/
   AlgebraicField()(i))**k2
sage: abs(aut_fact2 - f_rho.parent().aut_factor(A, z)) < 1e-12
True
sage: cor_term = (4 * G.n() / (G.n()-2) * A.c() * (A.c()**z+A.d())) / (2*pi*i).
   n(1000) * G.lam()
sage: aut_fact*f(z) + cor_term*aut_fact2*f_rho(z)
-15.9978074989... - 29.2775758341...*I
sage: f.parent().default_num_prec(1000)
sage: f.parent().default_prec(300)
sage: (f.q_expansion_fixed_d().polynomial())(exp((2*pi*i).n(1000)*z/G.lam()))
    # long time
0.9660243864186729677809436... - 0.013889469942995530807311503...*I
sage: (f.q_expansion_fixed_d().polynomial())(exp((2*pi*i).n(1000)*az/G.lam()))
    # long time
-15.997807498958825352887040... - 29.277575834123246063432206...*I
sage: MR = QuasiMeromorphicModularFormsRing(n=infinity, red_hom=True)
sage: f_i = MR.f_i().full_reduce()
sage: f_inf = MR.f_inf().full_reduce()
sage: E2 = MR.E2().full_reduce()
sage: E4 = MR.E4().full_reduce()
sage: f_i(i)
0
sage: f_i(i + 1e-1000)
2.991...e-12 - 3.048...e-1000*I
sage: f_inf(infinity)
0
sage: z = -1/(-1/(2*i+30)-1)
sage: E4(z, prec=15)
804.6722034... + 211.9278206...*I
sage: E4(z, prec=30, num_prec=100)  # long time
803.928382417... + 211.889914044...*I
sage: E2(z)
2.438455612... - 39.48442265...*I
sage: E2(z, prec=30, num_prec=100)  # long time
2.43968197227756036957475... - 39.4842637577742677851431...*I
sage: (E2^2-E4)(z)
-2265.442515... - 380.3197877...*I
sage: (E2^2-E4)(z, prec=30, num_prec=100)  # long time
-2265.44251550679807447320... - 380.319787790548788238792...*I
sage: (E2^2-E4)(infinity)
0
sage: (1/(E2^2-E4))(infinity)
+Infinity
sage: ((E2^2-E4)/f_inf)(infinity)
-1/(2*d)

(continues on next page)
sage: G = HeckeTriangleGroup(n=Infinity)
sage: z = AlgebraicField()(1/10+13/10*I)
sage: A = G.V(4)
sage: S = G.S()
sage: T = G.T()
sage: A == (T*S)**3*T
True
sage: az = A.acton(z)
sage: az == (A[0,0]*z + A[0,1]) / (A[1,0]*z + A[1,1])
True
sage: f_i(z)
0.620885340917559158572271... - 0.121252549240996430425967...*I
sage: f_i(az)
6.1033141975198186745017... + 20.4267859728657976382684...*I
sage: k = f_i.weight()
sage: aut_fact = f_i.ep()^3 * ((T*S)**2*T).acton(z)/AlgebraicField()(i)**k * ((T*S)*T).acton(z)/AlgebraicField()(i)**k * (T.acton(z)/AlgebraicField()(i))**k
sage: abs(aut_fact - f_i.parent().aut_factor(A, z)) < 1e-12
True
sage: aut_fact * f_i(z)
6.1033141975198186745017... + 20.4267859728657976382684...*I
sage: f_i.parent().default_num_prec(1000)
sage: f_i.parent().default_prec(300)
sage: (f_i.q_expansion_fixed_d().polynomial())(exp((2*pi*i).n(1000)*z/G.lam()))
# long time
0.620885340917559158572271... - 0.121252549240996430425967...*I
sage: (f_i.q_expansion_fixed_d().polynomial())(exp((2*pi*i).n(1000)*az/G.lam()))
# long time
6.1033141975198186745017... + 20.4267859728657976382684...*I
sage: f = f_i*E2
sage: f(z)
0.5349190275... - 0.1322370856...*I
sage: f(az)
-140.4711702... + 469.0793692...*I
sage: k2 = f_i.weight()
sage: aut_fact2 = f_i.ep() * ((T*S)**2*T).acton(z)/AlgebraicField()(i)**k2 * (T.acton(z)/AlgebraicField()(i))**k2
sage: abs(aut_fact2 - f_i.parent().aut_factor(A, z)) < 1e-12
True
sage: cor_term = (4 * A.c() * (A.c()^z+A.d())) / (2*pi*i).n(1000) * G.lam()
sage: aut_fact*f(z) + cor_term*aut_fact2*f_i(z)
It is possible to evaluate at points of HyperbolicPlane():

\[
\text{sage: } p = \text{HyperbolicPlane().PD().get_point(-I/2)}
\]
\[
\text{sage: } \text{bool}(p.to_model('UHP').coordinates() == I/3)
\]
True
\[
\text{sage: } E4(p) == E4(I/3)
\]
True
\[
\text{sage: } p = \text{HyperbolicPlane().PD().get_point(I)}
\]
\[
\text{sage: } \text{f_inf}(p, \text{check=True}) == 0
\]
True
\[
\text{sage: } (1/(E2^2-E4))(p) == \text{infinity}
\]
True

full_reduce()

Convert self into its reduced parent.

EXAMPLES:

\[
\text{sage: from sage.modular.modform_hecketrianglegraded_ring import } \text{QuasiMeromorphicModularFormsRing}
\]
\[
\text{sage: Delta = QuasiMeromorphicModularFormsRing().Delta()}
\]
\[
\text{sage: Delta, } f_\rho^3 d - f_i^2 d
\]
\[
\text{sage: Delta.full_reduce()}
\]
\[
q - 24q^2 + 252q^3 - 1472q^4 + O(q^5)
\]
\[
\text{sage: Delta.full_reduce().parent()} == \text{Delta.reduced_parent()}
\]
True
\[
\text{sage: QuasiMeromorphicModularFormsRing().Delta().full_reduce().parent()}
\]
\[
\text{CuspForms(n=3, k=12, ep=1) over Integer Ring}
\]

group()

Return the (Hecke triangle) group of self.parent().

EXAMPLES:

\[
\text{sage: from sage.modular.modform_hecketriangle.space import } \text{ModularForms}
\]
\[
\text{sage: ModularForms(n=12, k=4).E4().group()}
\]
\[
\text{Hecke triangle group for } n = 12
\]

hecke_n()

Return the parameter n of the (Hecke triangle) group of self.parent().

2.5. Elements of graded rings of modular forms for Hecke triangle groups 185
EXAMPLES:

```python
sage: from sage.modular.modform_hecketriangle.space import ModularForms
sage: ModularForms(n=12, k=6).E6().hecke_n()
12
```

**is_cuspidal()**

Return whether self is cuspidal in the sense that self is holomorphic and f_inf divides the numerator.

EXAMPLES:

```python
sage: from sage.modular.modform_hecketriangle.graded_ring import QuasiModularFormsRing
sage: from sage.modular.modform_hecketriangle.space import QuasiModularForms
sage: x,y,z,d = var("x,y,z,d")
sage: QuasiModularFormsRing(n=5)(y^3-z^5).is_cuspidal()
False
sage: QuasiModularFormsRing(n=5)(z*x^5-z*y^2).is_cuspidal()
True
sage: QuasiModularForms(n=18).Delta().is_cuspidal()
True
sage: QuasiModularForms(n=18).f_rho().is_cuspidal()
False
sage: QuasiModularForms(n=Infinity).f_inf().is_cuspidal()
False
sage: QuasiModularForms(n=Infinity).Delta().is_cuspidal()
True
```

**is_holomorphic()**

Return whether self is holomorphic in the sense that the denominator of self is constant.

EXAMPLES:

```python
sage: from sage.modular.modform_hecketriangle.graded_ring import QuasiMeromorphicModularFormsRing
sage: from sage.modular.modform_hecketriangle.space import QuasiMeromorphicModularForms
sage: x,y,z,d = var("x,y,z,d")
sage: QuasiMeromorphicModularFormsRing(n=5)((y^3-z^5)/(x^5-y^2)+y-d).is_holomorphic()
False
sage: QuasiMeromorphicModularFormsRing(n=5)(x^2+y-d-z).is_holomorphic()
True
sage: QuasiMeromorphicModularForms(n=18).J_inv().is_holomorphic()
False
sage: QuasiMeromorphicModularForms(n=18).f_i().is_holomorphic()
True
sage: QuasiMeromorphicModularForms(n=Infinity).f_inf().is_holomorphic()
True
```

**is_homogeneous()**

Return whether self is homogeneous.

EXAMPLES:
sage: from sage.modular.modform_hecketriangle.graded_ring import QuasiModularFormsRing
sage: QuasiModularFormsRing(n=12).Delta().is_homogeneous()
True
sage: QuasiModularFormsRing(n=12).Delta().parent().is_homogeneous()
False
sage: x,y,z,d=var("x,y,z,d")
sage: QuasiModularFormsRing(n=12)(x^3+y^2+z+d).is_homogeneous()
False
sage: QuasiModularFormsRing(n=infinity)(x*(x-y^2)+y^4).is_homogeneous()
True

is_modular()
Return whether self (resp. its homogeneous components) transform like modular forms.

EXAMPLES:

sage: from sage.modular.modform_hecketriangle.graded_ring import QuasiMeromorphicModularFormsRing
sage: from sage.modular.modform_hecketriangle.space import QuasiMeromorphicModularForms
sage: x,y,z,d = var("x,y,z,d")
sage: QuasiMeromorphicModularFormsRing(n=5)(x/(x^5-y^2)+z).is_weakly_holomorphic()
True
sage: QuasiMeromorphicModularFormsRing(n=5)(x^2+y/x-d).is_weakly_holomorphic()
False
sage: QuasiMeromorphicModularForms(n=18).J_inv().is_weakly_holomorphic()
True
sage: QuasiMeromorphicModularForms(n=infinity, k=-4)(1/x).is_weakly_holomorphic()
True

is_weakly_holomorphic()
Return whether self is weakly holomorphic in the sense that: self has at most a power of \( f_\infty \) in its denominator.

EXAMPLES:
sage: QuasiMeromorphicModularForms(n=infinity, k=-2)(1/y).is_weakly_holomorphic()  
False

\texttt{is\_zero()}

Return whether \texttt{self} is the zero function.

\textbf{EXAMPLES:}

\begin{verbatim}
sage: from sage.modular.modform_hecketriangle.graded_ring import QuasiModularFormsRing  
sage: from sage.modular.modform_hecketriangle.space import QuasiModularForms  
sage: x,y,z,d = var("x,y,z,d")  
sage: QuasiModularFormsRing(n=5)(1).is_zero()  
False  
sage: QuasiModularFormsRing(n=5)(0).is_zero()  
True  
sage: QuasiModularForms(n=18).zero().is_zero()  
True  
sage: QuasiModularForms(n=18).Delta().is_zero()  
False  
sage: QuasiModularForms(n=infinity).f_rho().is_zero()  
False
\end{verbatim}

\texttt{numerator()}

Return the numerator of \texttt{self}.

I.e. the (properly reduced) new form corresponding to the numerator of \texttt{self.rat()}.

Note that the parent of \texttt{self} might (probably will) change.

\textbf{EXAMPLES:}

\begin{verbatim}
sage: from sage.modular.modform_hecketriangle.graded_ring import QuasiMeromorphicModularFormsRing  
sage: from sage.modular.modform_hecketriangle.space import QuasiMeromorphicModularForms  
sage: x,y,z,d = var("x,y,z,d")  
sage: QuasiMeromorphicModularFormsRing(n=5)((y^3-z^5)/(x^5-y^2)+y-d).numerator()  
f_rhoo^5*f_i - f_rhoo^5*d - E2^5 + f_i^2*d  
sage: QuasiMeromorphicModularFormsRing(n=5)((y^3-z^5)/(x^5-y^2)+y-d).numerator().parent()  
QuasiModularFormsRing(n=5) over Integer Ring  
sage: QuasiMeromorphicModularForms(n=5, k=-2, ep=-1)(x/y).numerator()  
1 + 7/(100*d)*q + 21/(16000000*d^2)*q^2 + 1043/(1920000000*d^3)*q^3 + 45479/(12288000000000*d^4)*q^4 + O(q^5)  
sage: QuasiMeromorphicModularForms(n=5, k=4/3, ep=1) over Integer Ring
\end{verbatim}
order_at(tau=+Infinity)

Return the (overall) order of self at tau if easily possible: Namely if tau is infinity or congruent to i resp. rho.

It is possible to determine the order of points from HyperbolicPlane(). In this case the coordinates of the upper half plane model are used.

If self is homogeneous and modular then the rational function self.rat() is used. Otherwise only tau=infinity is supported by using the Fourier expansion with increasing precision (until the order can be determined).

The function is mainly used to be able to work with the correct precision for Laurent series.

**Note:** For quasi forms one cannot deduce the analytic type from this order at infinity since the analytic order is defined by the behavior on each quasi part and not by their linear combination.

**EXAMPLES:**

```python
sage: from sage.modular.modform_hecketriangle.graded_ring import _
    QuasiMeromorphicModularFormsRing
sage: MR = QuasiMeromorphicModularFormsRing(red_hom=True)

sage: (MR.Delta()^3).order_at(infinity)
3
sage: (MR.E2()).order_at(infinity)
0
sage: (MR.J_inv()^2).order_at(infinity)
-2

sage: x,y,z,d = MR.pol_ring().gens()
sage: el = MR((z^3-y)/(x^3-y^2)).full_reduce()
sage: el
108*q + 11664*q^2 + 502848*q^3 + 12010464*q^4 + O(q^5)
sage: el.order_at(infinity)
1

sage: el.parent()
QuasiWeakModularForms(n=3, k=0, ep=1) over Integer Ring
sage: el.is_holomorphic()
False
sage: MR((z-y)^2+(x-y)^3).order_at(infinity)
2
sage: MR((x-y)^10).order_at(infinity)
10
sage: MR.zero().order_at(infinity)
+Infinity
sage: (MR(x*y^2)/MR.J_inv()).order_at(i)
2
sage: (MR(x*y^2)/MR.J_inv()).order_at(MR.group().rho())
-2

sage: MR = QuasiMeromorphicModularFormsRing(n=infinity, red_hom=True)

sage: (MR.Delta()^3*MR.E4()).order_at(infinity)
3
sage: (MR.E2()).order_at(infinity)
0
sage: (MR.J_inv()^2/MR.E4()).order_at(infinity)
```

(continues on next page)
sage: el = MR((z^3-x*y)^2/(x^2*(x-y^2))).full_reduce()
sage: el
4*q - 304*q^2 + 8128*q^3 - 106144*q^4 + O(q^5)
sage: el.order_at(infinity)
1
sage: el.parent()
QuasiWeakModularForms(n=+Infinity, k=0, ep=1) over Integer Ring
sage: el.is_holomorphic()
False
sage: MR((z-x)^2+(x-y)^3).order_at(infinity)
2
sage: MR((x-y)^10).order_at(infinity)
10
sage: MR.zero().order_at(infinity)
+Infinity

\textbf{q\_expansion}(\textit{prec=None, fix\_d=False, d\_num\_prec=None, fix\_prec=False})

Returns the Fourier expansion of self.

INPUT:

- \textbf{prec} – An integer, the desired output precision $O(q^{\text{prec}})$.
  Default: None in which case the default precision of self.parent() is used.

- \textbf{fix\_d} – If False (default) a formal parameter is used for $d$.
  If True then the numerical value of $d$ is used (resp. an exact value if the group is arithmetic).
  Otherwise the given value is used for $d$.

- \textbf{d\_num\_prec} – The precision to be used if a numerical value for $d$ is substituted.
  Default: None in which case the default numerical precision of self.parent() is used.

- \textbf{fix\_prec} – If fix\_prec is not False (default)
  then the precision of the MFSeriesConstructor is increased such that the output has exactly the
  specified precision $O(q^{\text{prec}})$.

OUTPUT:

The Fourier expansion of self as a FormalPowerSeries or FormalLaurentSeries.

EXAMPLES:
Modular Forms, Release 10.0

```python
sage: from sage.modular.modform_hecketriangle.graded_ring import _
˓→WeakModularFormsRing, QuasiModularFormsRing
sage: j_inv = WeakModularFormsRing(red_hom=True).j_inv()
sage: j_inv.q_expansion(prec=3)
q^-1 + 31/(72*d) + 1823/(27648*d^2)*q + 10495/(2519424*d^3)*q^2 + O(q^3)

sage: E2 = QuasiModularFormsRing(n=5, red_hom=True).E2()
sage: E2.q_expansion(prec=3)
1 - 9/(200*d)*q - 369/(320000*d^2)*q^2 + O(q^3)
sage: E2.q_expansion(prec=3, fix_d=1)
1 - 9/200*q - 369/320000*q^2 + O(q^3)

sage: E6 = WeakModularFormsRing(n=5, red_hom=True).E6().full_reduce()
sage: Delta = WeakModularFormsRing(n=5, red_hom=True).Delta().full_reduce()
sage: E6.q_expansion(prec=3).prec() == 3
True
sage: (Delta/(E2^3-E6)).q_expansion(prec=3).prec() == 3
True
sage: (Delta/(E2^3-E6)^3).q_expansion(prec=3).prec() == 3
True
sage: ((E^2-E6)/Delta^2).q_expansion(prec=3).prec() == 3
True
sage: ((E^2-E6)^3/Delta).q_expansion(prec=3).prec() == 3
True

sage: x,y = var("x,y")
sage: el = WeakModularFormsRing()((x+1)/(x^3-y^2))
sage: el.q_expansion(prec=2, fix_prec = True)
2*d*q^-1 + O(1)
sage: el.q_expansion(prec=2)
2*d*q^-1 + 1/6 + 119/(41472*d)*q + O(q^2)

sage: j_inv = WeakModularFormsRing(n=infinity, red_hom=True).j_inv()
sage: j_inv.q_expansion(prec=3)
q^-1 + 3/(8*d) + 69/(1024*d^2)*q + 1/(128*d^3)*q^2 + O(q^3)

sage: E2 = QuasiModularFormsRing(n=infinity, red_hom=True).E2()
sage: E2.q_expansion(prec=3)
1 - 1/(8*d)*q - 1/(512*d^2)*q^2 + O(q^3)
sage: E2.q_expansion(prec=3, fix_d=1)
1 - 1/8*q - 1/512*q^2 + O(q^3)

sage: E4 = WeakModularFormsRing(n=infinity, red_hom=True).E4().full_reduce()
sage: Delta = WeakModularFormsRing(n=infinity, red_hom=True).Delta().full_reduce()
sage: E4.q_expansion(prec=3).prec() == 3
True
sage: (Delta/(E2^2-E4)).q_expansion(prec=3).prec() == 3
True
sage: (Delta/(E2^2-E4)^3).q_expansion(prec=3).prec() == 3
True
sage: ((E^2-E4)/Delta^2).q_expansion(prec=3).prec() == 3
True
```

(continues on next page)
sage: ((E2^2-E4)^3/Delta).q_expansion(prec=3).prec() == 3
True

sage: x,y = var("x,y")
sage: el = WeakModularFormsRing(n=infinity)((x+1)/(x-y^2))
sage: el.q_expansion(prec=2, fix_prec = True)
2*d*q^-1 + 0(1)
sage: el.q_expansion(prec=2)
2*d*q^-1 + 1/2 + 39/(512*d)*q + O(q^2)

q_expansion_fixed_d(prec=None, d_num_prec=None, fix_prec=False)

Returns the Fourier expansion of self. The numerical (or exact) value for d is substituted.

INPUT:

• prec – An integer, the desired output precision O(q^prec).
  Default: None in which case the default precision of self.parent() is used.

• d_num_prec – The precision to be used if a numerical value for d is substituted.
  Default: None in which case the default numerical precision of self.parent() is used.

• fix_prec – If fix_prec is not False (default) then the precision of the MFseriesConstructor
  is increased such that the output has exactly the specified precision O(q^prec).

OUTPUT:

The Fourier expansion of self as a FormalPowerSeries or FormalLaurentSeries.

EXAMPLES:

sage: from sage.modular.modform_hecketriangle.graded_ring import ...
         WeakModularFormsRing, QuasiModularFormsRing
sage: j_inv = WeakModularFormsRing(red_hom=True).j_inv()
sage: j_inv.q_expansion_fixed_d(prec=3)
q^-1 + 744 + 196884*q + 21493760*q^2 + O(q^3)

sage: E2 = QuasiModularFormsRing(n=5, red_hom=True).E2()
sage: E2.q_expansion_fixed_d(prec=3)
1.00000000000... - 8.380956565426...*q - 23.18584547617...*q^2 + O(q^3)

sage: x,y = var("x,y")
sage: WeakModularFormsRing((x+1)/(x^3-y^2)).q_expansion_fixed_d(prec=2, fix_
  prec = True)
1/864*q^-1 + 0(1)
sage: WeakModularFormsRing((x+1)/(x^3-y^2)).q_expansion_fixed_d(prec=2)
1/864*q^-1 + 1/6 + 119/24*q + O(q^2)

sage: j_inv = WeakModularFormsRing(n=infinity, red_hom=True).j_inv()
sage: j_inv.q_expansion_fixed_d(prec=3)
q^-1 + 24 + 276*q + 2048*q^2 + O(q^3)

sage: E2 = QuasiModularFormsRing(n=infinity, red_hom=True).E2()
sage: E2.q_expansion_fixed_d(prec=3)
-8*q - 8*q^2 + O(q^3)
sage: x, y = var("x, y")
sage: WeakModularFormsRing(n=infinity)((x+1)/(x-y^2)).q_expansion_fixed_d(prec=2, fix_prec=True)
1/32*q^-1 + 0(1)
sage: WeakModularFormsRing(n=infinity)((x+1)/(x-y^2)).q_expansion_fixed_d(prec=2)
1/32*q^-1 + 1/2 + 39/8*q + 0(q^2)

sage: (WeakModularFormsRing(n=14).j_inv()^3).q_expansion_fixed_d(prec=2)
2.93373093...e-6*q^-3 + 0.0002320999814...*q^-2 + 0.00913529265...*q^-1 + 0.
→+ 2292916854... + 4.303583833...*q + 0(q^2)

q_expansion_vector(min_exp=None, max_exp=None, prec=None, **kwargs)

Return (part of) the Laurent series expansion of self as a vector.

INPUT:

- **min_exp** – An integer, specifying the first coefficient to be
  used for the vector. Default: None, meaning that the first non-trivial coefficient is used.

- **max_exp** – An integer, specifying the last coefficient to be
  used for the vector. Default: None, meaning that the default precision + 1 is used.

- **prec** – An integer, specifying the precision of the underlying
  Laurent series. Default: None, meaning that max_exp + 1 is used.

OUTPUT:

A vector of size max_exp - min_exp over the coefficient ring of self, determined by the corresponding
Laurent series coefficients.

EXAMPLES:

sage: from sage.modular.modformhecketriangle.graded_ring import..
→WeakModularFormsRing
sage: f = WeakModularFormsRing(red_hom=True).j_inv()^3
sage: f.q_expansion(prec=3)
q^-3 + 31/(24*d)*q^-2 + 20845/(27648*d^2)*q^-1 + 705834/(26873856*d^3) +
→30098784355/(495338913792*d^4)*q + 175372747465/(17832200896512*d^5)*q^2 +
→O(q^3)
sage: v = f.q_expansion_vector(max_exp=1, prec=3)
sage: v
(1, 31/(24*d), 20845/(27648*d^2), 705834/(26873856*d^3), 30098784355/
→(495338913792*d^4))
sage: v.parent()
Vector space of dimension 5 over Fraction Field of Univariate Polynomial Ring
→in d over Integer Ring
sage: f.q_expansion_vector(min_exp=1, max_exp=2)
(30098784355/(495338913792*d^4), 175372747465/(17832200896512*d^5))
sage: f.q_expansion_vector(min_exp=1, max_exp=2, fix_d=True)
(541778118390, 151522053809760)
sage: f = WeakModularFormsRing(n=infinity, red_hom=True).j_inv()^3
sage: f.q_expansion_fixed_d(prec=3)
\[ q^{-3} + 72q^{-2} + 2556q^{-1} + 59712 + 1033974q + 14175648q^2 + O(q^3) \]

```
sage: v = f.q_expansion_vector(max_exp=1, prec=3, fix_d=True)
sage: v
(1, 72, 2556, 59712, 1033974)
sage: v.parent()
Vector space of dimension 5 over Rational Field
```

```
sage: f.q_expansion_vector(min_exp=1, max_exp=2)
(516987/(8388608*d^4), 442989/(33554432*d^5))
```

rat()

Return the rational function representing self.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.graded_ring import...
˓→ModularFormsRing
sage: ModularFormsRing(n=12).Delta().rat()
x^30*d - x^18*y^2*d
```

reduce(force=False)

In case self.parent().has_reduce_hom() == True (or force==True) and self is homogeneous the converted element lying in the corresponding homogeneous_part is returned.

Otherwise self is returned.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.graded_ring import...
˓→ModularFormsRing
sage: E2 = ModularFormsRing(n=7).E2().reduce()
sage: E2.parent()
QuasiModularFormsRing(n=7) over Integer Ring
sage: E2 = ModularFormsRing(n=7, red_hom=True).E2().reduce()
sage: E2.parent()
QuasiModularForms(n=7, k=2, ep=-1) over Integer Ring
```

```
sage: ModularFormsRing(n=7)(x+y^2).reduce(force=True)
64*q - 512*q^2 + 1792*q^3 - 4096*q^4 + O(q^5)
```

reduced_parent()

Return the space with the analytic type of self. If self is homogeneous the corresponding FormsSpace is returned.

I.e. return the smallest known ambient space of self.

EXAMPLES:
sage: from sage.modular.modform_hecketriangle.graded_ring import...
    \rightarrow \text{QuasiMeromorphicModularFormsRing}
sage: Delta = QuasiMeromorphicModularFormsRing(n=7).Delta()
sage: Delta.parent()
QuasiMeromorphicModularFormsRing(n=7) over Integer Ring
sage: Delta.reduced_parent()
CuspForms(n=7, k=12, ep=1) over Integer Ring
sage: el = QuasiMeromorphicModularFormsRing()(x+1)
sage: el.parent()
QuasiMeromorphicModularFormsRing(n=3) over Integer Ring
sage: el.reduced_parent()
ModularFormsRing(n=3) over Integer Ring
sage: y = var("y")
sage: QuasiMeromorphicModularFormsRing(n=infinity)(x-y^2).reduced_parent()
ModularForms(n=+Infinity, k=4, ep=1) over Integer Ring
sage: QuasiMeromorphicModularFormsRing(n=infinity)(x*(x-y^2)).reduced_parent()
CuspForms(n=+Infinity, k=8, ep=1) over Integer Ring

\textbf{serre_derivative()}

Return the Serre derivative of \texttt{self}.

Note that the parent might (probably will) change. However a modular element is returned if \texttt{self} was already modular.

If parent.has_reduce_hom() == True then the result is reduced to be an element of the corresponding forms space if possible.

In particular this is the case if \texttt{self} is a (homogeneous) element of a forms space.

\textbf{EXAMPLES:}

sage: from sage.modular.modform_hecketriangle.graded_ring import...
    \rightarrow \text{QuasiMeromorphicModularFormsRing}
sage: MR = QuasiMeromorphicModularFormsRing(n=7, red_hom=True)
sage: n = MR.hecke_n()
sage: Delta = MR.Delta().full_reduce()
sage: E2 = MR.E2().full_reduce()
sage: E4 = MR.E4().full_reduce()
sage: E6 = MR.E6().full_reduce()
sage: f_rho = MR.f_rho().full_reduce()
sage: f_i = MR.f_i().full_reduce()
sage: f_inf = MR.f_inf().full_reduce()

sage: f_rho.serre_derivative() == -1/n * f_i
True
sage: f_i.serre_derivative() == -1/2 * E4 * f_rho
True
sage: f_inf.serre_derivative() == 0
True
sage: E2.serre_derivative() == -(n-2)/(4*n) * (E2^2 + E4)
True
sage: E4.serre_derivative() == -(n-2)/n * E6
True
sage: E6.serre_derivative() == -1/2 * E4^2 - (n-3)/n * E6^2 / E4

(continues on next page)
True

\texttt{sage}: \texttt{E6.serre_derivative().parent()}
\texttt{ModularForms(n=7, k=8, ep=1) over Integer Ring}

\texttt{sage}: \texttt{MR = QuasiMeromorphicModularFormsRing(n=infinity, red_hom=True)}
\texttt{sage}: \texttt{Delta = MR.Delta().full_reduce()}
\texttt{sage}: \texttt{E2 = MR.E2().full_reduce()}
\texttt{sage}: \texttt{E4 = MR.E4().full_reduce()}
\texttt{sage}: \texttt{E6 = MR.E6().full_reduce()}
\texttt{sage}: \texttt{f_i = MR.f_i().full_reduce()}
\texttt{sage}: \texttt{f_inf = MR.f_inf().full_reduce()}

\texttt{sage}: \texttt{E4.serre_derivative() == -E4 * f_i}
\texttt{True}
\texttt{sage}: \texttt{f_i.serre_derivative() == -1/2 * E4}
\texttt{True}
\texttt{sage}: \texttt{f_inf.serre_derivative() == 0}
\texttt{True}
\texttt{sage}: \texttt{E2.serre_derivative() == -1/4 * (E2^2 + E4)}
\texttt{True}
\texttt{sage}: \texttt{E4.serre_derivative() == -E6}
\texttt{True}
\texttt{sage}: \texttt{E6.serre_derivative() == -1/2 * E4^2 - E6^2 / E4}
\texttt{True}
\texttt{sage}: \texttt{E6.serre_derivative().parent()}
\texttt{ModularForms(n=+Infinity, k=8, ep=1) over Integer Ring}

\texttt{sqrt()}

Return the square root of \texttt{self} if it exists.

I.e. the element corresponding to \texttt{sqrt(self.rat())}.

Whether this works or not depends on whether \texttt{sqrt(self.rat())} works and coerces into \texttt{self.parent().rat_field()}.

Note that the parent might (probably will) change.

If \texttt{parent.has_reduce_hom() == True} then the result is reduced to be an element of the corresponding forms space if possible.

In particular this is the case if \texttt{self} is a (homogeneous) element of a forms space.

\textbf{EXAMPLES:}

\texttt{sage}: \texttt{from sage.modular.modform_hecketriangle.space import QuasiModularForms}
\texttt{sage}: \texttt{E2 = QuasiModularForms(k=2, ep=-1).E2()}
\texttt{sage}: \texttt{(E2^2).sqrt()}
\texttt{1 - 24*q - 72*q^2 - 96*q^3 - 168*q^4 + O(q^5)}
\texttt{sage}: \texttt{(E2^2).sqrt() == E2}
\texttt{True}

\texttt{weight()}

Return the weight of \texttt{self}.

\textbf{EXAMPLES:}
**2.6 Constructor for spaces of modular forms for Hecke triangle groups based on a type**

**AUTHORS:**

- Jonas Jermann (2013): initial version

sage.modular.modform_hecketriangle.constructor.FormsRing(analytic_type, group=3, base_ring=Integer Ring, red_hom=False)

Return the FormsRing with the given analytic_type, group base_ring and variable red_hom.

**INPUT:**

- **analytic_type** – An element of AnalyticType() describing the analytic type of the space.
- **group** – The index of the (Hecke triangle) group of the space (default: 3').
- **base_ring** – The base ring of the space (default: ZZ).
- **red_hom** – The (boolean= variable red_hom of the space (default: False).

For the variables group, base_ring, red_hom the same arguments as for the class FormsRing_abstract can be used. The variables will then be put in canonical form.

**OUTPUT:**

The FormsRing with the given properties.

**EXAMPLES:**

sage: from sage.modular.modform_hecketriangle.constructor import FormsRing

sage: FormsRing("cusp", group=5, base_ring=CC)
CuspFormsRing(n=5) over Complex Field with 53 bits of precision

sage: FormsRing("holo")
ModularFormsRing(n=3) over Integer Ring

sage: FormsRing("weak", group=6, base_ring=ZZ, red_hom=True)
WeakModularFormsRing(n=6) over Integer Ring

(continues on next page)
sage: FormsRing("mero", group=7, base_ring=ZZ)
MeromorphicModularFormsRing(n=7) over Integer Ring

sage: FormsRing(\["quasi", "cusp"\], group=5, base_ring=CC)
QuasiCuspFormsRing(n=5) over Complex Field with 53 bits of precision

sage: FormsRing(\["quasi", "holo"\])
QuasiModularFormsRing(n=3) over Integer Ring

sage: FormsRing(\["quasi", "weak"\], group=6, base_ring=ZZ, red_hom=True)
QuasiWeakModularFormsRing(n=6) over Integer Ring

sage: FormsRing(\["quasi", "mero"\], group=7, base_ring=ZZ, red_hom=True)
QuasiMeromorphicModularFormsRing(n=7) over Integer Ring

sage: FormsRing(\["quasi", "cusp"\], group=infinity)
QuasiCuspFormsRing(n=+Infinity) over Integer Ring

sage.modular.modform_hecketriangle.constructor.FormsSpace(\analytic_type, group=3, base_ring=Integer Ring, k=0, ep=None\)

Return the FormsSpace with the given analytic_type, group base_ring and degree (k, ep).

**INPUT:**

- **analytic_type** – An element of AnalyticType() describing the analytic type of the space.
- **group** – The index of the (Hecke triangle) group of the space (default: 3).
- **base_ring** – The base ring of the space (default: ZZ).
- **k** – The weight of the space, a rational number (default: 0).
- **ep** – The multiplier of the space, 1, −1 or None (in case ep should be determined from k). Default: None.

For the variables group, base_ring, k, ep the same arguments as for the class FormsSpace_abstract can be used. The variables will then be put in canonical form. In particular the multiplier ep is calculated as usual from k if ep == None.

**OUTPUT:**

The FormsSpace with the given properties.

**EXAMPLES:**

```python
sage: from sage.modular.modform_hecketriangle.constructor import FormsSpace
sage: FormsSpace([])
ZeroForms(n=3, k=0, ep=1) over Integer Ring
sage: FormsSpace(["quasi"]\) # not implemented

sage: FormsSpace("cusp", group=5, base_ring=CC, k=12, ep=1)
CuspForms(n=5, k=12, ep=1) over Complex Field with 53 bits of precision
```
sage: FormsSpace("holo")
ModularForms(n=3, k=0, ep=1) over Integer Ring

sage: FormsSpace("weak", group=6, base_ring=ZZ, k=0, ep=-1)
WeakModularForms(n=6, k=0, ep=-1) over Integer Ring

sage: FormsSpace("mero", group=7, base_ring=ZZ, k=2, ep=-1)
MeromorphicModularForms(n=7, k=2, ep=-1) over Integer Ring

sage: FormsSpace(["quasi", "cusp"], group=5, base_ring=CC, k=12, ep=1)
QuasiCuspForms(n=5, k=12, ep=1) over Complex Field with 53 bits of precision

sage: FormsSpace(["quasi", "holo"])
QuasiModularForms(n=3, k=0, ep=1) over Integer Ring

sage: FormsSpace(["quasi", "weak"], group=6, base_ring=ZZ, k=0, ep=-1)
QuasiWeakModularForms(n=6, k=0, ep=-1) over Integer Ring

sage: FormsSpace(["quasi", "mero"], group=7, base_ring=ZZ, k=2, ep=-1)
QuasiMeromorphicModularForms(n=7, k=2, ep=-1) over Integer Ring

sage: FormsSpace(["quasi", "cusp"], group=8, base_ring=ZZ, k=2, ep=-1)
QuasiCuspForms(n=+Infinity, k=2, ep=-1) over Integer Ring

sage.modular.modform_hecketriangle.constructor.rational_type(f, n=3, base_ring=Integer Ring)

Return the basic analytic properties that can be determined directly from the specified rational function \( f \) which is interpreted as a representation of an element of a FormsRing for the Hecke Triangle group with parameter \( n \) and the specified base\_ring.

In particular the following degree of the generators is assumed:

\[
\begin{align*}
\deg(1) & := (0, 1) \\
\deg(x) & := (4/(n-2), 1) \\
\deg(y) & := (2n/(n-2), -1) \\
\deg(z) & := (2, -1)
\end{align*}
\]

The meaning of homogeneous elements changes accordingly.

INPUT:

- \( f \) – A rational function in \( x, y, z, d \) over base\_ring.
- \( n \) – An integer greater or equal to 3 corresponding to the HeckeTriangleGroup with that parameter (default: 3).
- \( \text{base}\_\text{ring} \) – The base ring of the corresponding forms ring, resp. polynomial ring (default: ZZ).

OUTPUT:

A tuple (elem, homo, k, ep, analytic\_type) describing the basic analytic properties of \( f \) (with the interpretation indicated above).

- \( \text{elem} \) – True if \( f \) has a homogeneous denominator.
- \( \text{homo} \) – True if \( f \) also has a homogeneous numerator.
- \( k \) – None if \( f \) is not homogeneous, otherwise the weight of \( f \) (which is the first component of its degree).
- \( ep \) – None if \( f \) is not homogeneous, otherwise the multiplier of \( f \) (which is the second component of its degree)
• **analytic_type** – The AnalyticType of \( f \).

For the zero function the degree \((0, 1)\) is chosen.

This function is (heavily) used to determine the type of elements and to check if the element really is contained in its parent.

**EXAMPLES:**

```sage
def import rational_type
sage: (x,y,z,d) = var("x,y,z,d")
sage: rational_type(0, n=4)
(True, True, 0, 1, zero)
sage: rational_type(1, n=12)
(True, True, 0, 1, modular)
sage: rational_type(x^3 - y^2)
(True, True, 12, 1, cuspidal)
sage: rational_type(x * z, n=7)
(True, True, 14/5, -1, quasi modular)
sage: rational_type(1/(x^3 - y^2) + z/d)
(True, False, None, None, quasi weakly holomorphic modular)
sage: rational_type(x^3/(x^3 - y^2))
(True, True, 0, 1, weakly holomorphic modular)
sage: rational_type(1/(x + z))
(False, False, None, None, None)
sage: rational_type(1/x + 1/z)
(True, False, None, None, quasi meromorphic modular)
sage: rational_type(d/x, n=10)
(True, True, -1/2, 1, meromorphic modular)
sage: rational_type(1.1 * z * (x^8-y^2), n=8, base_ring=CC)
(True, True, 22/3, -1, quasi cuspidal)
sage: rational_type(x-y^2, n=infinity)
(True, True, 4, 1, modular)
sage: rational_type(x*(x-y^2), n=infinity)
(True, True, 8, 1, cuspidal)
sage: rational_type(1/x, n=infinity)
(True, True, -4, 1, weakly holomorphic modular)
```
2.7 Functor construction for all spaces

AUTHORS:

- Jonas Jermann (2013): initial version

**class** `sage.modular.modform_hecketriangle.functors.BaseFacade(ring)`

Bases: `Parent`, `UniqueRepresentation`

BaseFacade of a ring.

This class is used to distinguish the construction of constant elements (modular forms of weight 0) over the given ring and the construction of `FormsRing` or `FormsSpace` based on the `BaseFacade` of the given ring.

If that distinction was not made then ring elements couldn’t be considered as constant modular forms in e.g. binary operations. Instead the coercion model would assume that the ring element lies in the common parent of the ring element and e.g. a `FormsSpace` which would give the `FormsSpace` over the ring. However this is not correct, the `FormsSpace` might (and probably will) not even contain the (constant) ring element. Hence we use the `BaseFacade` to distinguish the two cases.

Since the `BaseFacade` of a ring embeds into that ring, a common base (resp. a coercion) between the two (or even a more general ring) can be found, namely the ring (not the `BaseFacade` of it).

**sage.modular.modform_hecketriangle.functors.ConstantFormsSpaceFunctor(group)**

Construction functor for the space of constant forms.

When determining a common parent between a ring and a forms ring or space this functor is first applied to the ring.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.functors import ...
˓→(ConstantFormsSpaceFunctor, FormsSpaceFunctor)
sage: ConstantFormsSpaceFunctor(4) == FormsSpaceFunctor("holo", 4, 0, 1)
True
sage: ConstantFormsSpaceFunctor(4)
ModularFormsFunctor(n=4, k=0, ep=1)
```

**class** `sage.modular.modform_hecketriangle.functors.FormsRingFunctor(analytic_type, group, red_hom)`

Bases: `ConstructionFunctor`

Construction functor for forms rings.

NOTE:

When the base ring is not a `BaseFacade` the functor is first merged with the `ConstantFormsSpaceFunctor`. This case occurs in the pushout constructions. (when trying to find a common parent between a forms ring and a ring which is not a `BaseFacade`).

\[ \text{AT} = \text{Analytic Type} \]

**AnalyticType**

alias of `AnalyticType`

**merge**(other)

Return the merged functor of `self` and `other`.

It is only possible to merge instances of `FormsSpaceFunctor` and `FormsRingFunctor`. Also only if they share the same group. An `FormsSubSpaceFunctors` is replaced by its ambient space functor.
The analytic type of the merged functor is the extension of the two analytic types of the functors. The `red_hom` parameter of the merged functor is the logical and of the two corresponding `red_hom` parameters (where a forms space is assumed to have it set to `True`).

Two `FormsSpaceFunctor` with different `(k, ep)` are merged to a corresponding `FormsRingFunctor`. Otherwise the corresponding (extended) `FormsSpaceFunctor` is returned.

A `FormsSpaceFunctor` and `FormsRingFunctor` are merged to a corresponding (extended) `FormsRingFunctor`.

Two `FormsRingFunctors` are merged to the corresponding (extended) `FormsRingFunctor`.

**EXAMPLES:**

```python
sage: from sage.modular.modform_hecketriangle.functors import...
˓→ (FormsSpaceFunctor, FormsRingFunctor)

sage: functor1 = FormsRingFunctor("mero", group=6, red_hom=True)
sage: functor2 = FormsRingFunctor(['quasi', 'cusp'], group=6, red_hom=False)
sage: functor3 = FormsSpaceFunctor("weak", group=6, k=0, ep=1)
sage: functor4 = FormsRingFunctor("mero", group=5, red_hom=True)

sage: functor1.merge(functor1) is functor1
True
sage: functor1.merge(functor4) is None
True
sage: functor1.merge(functor2)
QuasiMeromorphicModularFormsRingFunctor(n=6)

sage: functor1.merge(functor3)
MeromorphicModularFormsRingFunctor(n=6, red_hom=True)
```

```python
rank = 10
```
A FormsSpaceFunctor and FormsRingFunctor are merged to a corresponding (extended) FormsRingFunctor.

Two FormsRingFunctors are merged to the corresponding (extended) FormsRingFunctor.

**EXAMPLES:**

```python
sage: from sage.modular.modform_hecketriangle.functors import (FormsSpaceFunctor, FormsRingFunctor)

sage: functor1 = FormsSpaceFunctor("holo", group=5, k=0, ep=1)
sage: functor2 = FormsSpaceFunctor(["quasi", "cusp"], group=5, k=10/3, ep=-1)
sage: functor3 = FormsSpaceFunctor(["quasi", "mero"], group=5, k=0, ep=1)
sage: functor4 = FormsRingFunctor("cusp", group=5, red_hom=False)
sage: functor5 = FormsSpaceFunctor("holo", group=4, k=0, ep=1)

sage: functor1.merge(functor1) is functor1
True
sage: functor1.merge(functor5) is None
True
sage: functor1.merge(functor2)
QuasiModularFormsRingFunctor(n=5, red_hom=True)
sage: functor1.merge(functor3)
QuasiMeromorphicModularFormsFunctor(n=5, k=0, ep=1)
sage: functor1.merge(functor4)
ModularFormsRingFunctor(n=5)
```

**rank = 10**

```python
class sage.modular.modform_hecketriangle.functors.FormsSubSpaceFunctor(ambient_space_functor, generators)
    Bases: ConstructionFunctor

    Construction functor for forms sub spaces.

    **merge**(other)
    
    Return the merged functor of self and other.

    If other is a FormsSubSpaceFunctor then first the common ambient space functor is constructed by merging the two corresponding functors.

    If that ambient space functor is a FormsSpaceFunctor and the generators agree the corresponding FormsSubSpaceFunctor is returned.

    If other is not a FormsSubSpaceFunctor then self is merged as if it was its ambient space functor.

    **EXAMPLES:**

    ```python
    sage: from sage.modular.modform_hecketriangle.functors import (FormsSpaceFunctor, FormsSubSpaceFunctor)
    sage: from sage.modular.modform_hecketriangle.space import ModularForms
    sage: ambient_space = ModularForms(n=4, k=12, ep=1)
    sage: ambient_space_functor1 = FormsSpaceFunctor("holo", group=4, k=12, ep=1)
    sage: ambient_space_functor2 = FormsSpaceFunctor("cusp", group=4, k=12, ep=1)
    sage: ss_functor1 = FormsSubSpaceFunctor(ambient_space_functor1, [ambient_space.gen(0)])
    sage: ss_functor2 = FormsSubSpaceFunctor(ambient_space_functor2, [ambient_space.gen(0)])
    ```
    ```
    ```
sage: ss_functor3 = FormsSubSpaceFunctor(ambient_space_functor2, [2*ambient_space.gen(0)])
sage: merged_ambient = ambient_space_functor1.merge(ambient_space_functor2)
sage: merged_ambient
ModularFormsFunctor(n=4, k=12, ep=1)
sage: functor4 = FormsSpaceFunctor(["quasi", "cusp"], group=4, k=10, ep=-1)
sage: ss_functor1.merge(ss_functor1) is ss_functor1
True
sage: ss_functor1.merge(ss_functor2)
FormsSubSpaceFunctor with 2 generators for the ModularFormsFunctor(n=4, k=12, ep=1)
sage: ss_functor1.merge(ss_functor2) == FormsSubSpaceFunctor(merged_ambient, [ambient_space.gen(0), ambient_space.gen(0)])
True
sage: ss_functor1.merge(ss_functor3) == FormsSubSpaceFunctor(merged_ambient, [ambient_space.gen(0), 2*ambient_space.gen(0)])
True
sage: ss_functor1.merge(ambient_space_functor2) == merged_ambient
True
sage: ss_functor1.merge(functor4)
QuasiModularFormsRingFunctor(n=4, red_hom=True)

rank = 10

2.8 Hecke triangle groups

AUTHORS:

- Jonas Jermann (2013): initial version

class sage.modular.modform_hecketriangle.hecke_triangle_groups.HeckeTriangleGroup(n)
Bases: FinitelyGeneratedMatrixGroup_generic, UniqueRepresentation
Hecke triangle group (2, n, infinity).

Element

alias of HeckeTriangleGroupElement

I()

Return the identity element/matrix for self.

EXAMPLES:

sage: from sage.modular.modform_hecketriangle.hecke_triangle_groups import HeckeTriangleGroup
sage: HeckeTriangleGroup(10).I()
[1 0]
[0 1]
sage: HeckeTriangleGroup(10).I().parent()
Hecke triangle group for n = 10
S()

Return the generator of self corresponding to the conformal circle inversion.

EXAMPLES:

```python
sage: from sage.modular.modform_hecketriangle.hecke_triangle_groups import HeckeTriangleGroup
sage: HeckeTriangleGroup(3).S()
[ 0 -1]
[ 1 0]
sage: HeckeTriangleGroup(10).S()
[ 0 -1]
[ 1 0]
True
sage: HeckeTriangleGroup(10).S()^4 == HeckeTriangleGroup(10).I()
True

sage: HeckeTriangleGroup(10).S().parent()
Hecke triangle group for n = 10
```

T(m=1)

Return the element in self corresponding to the translation by m*lam().

INPUT:

- m – An integer, default: 1, namely the second generator of self.

EXAMPLES:

```python
sage: from sage.modular.modform_hecketriangle.hecke_triangle_groups import HeckeTriangleGroup
sage: HeckeTriangleGroup(3).T()
[1 1]
[0 1]
sage: HeckeTriangleGroup(10).T(-4)
[ 1 -4*lam]
[ 0 1]
sage: HeckeTriangleGroup(10).T().parent()
Hecke triangle group for n = 10
```

U()

Return an alternative generator of self instead of T. U stabilizes rho and has order 2*n().

If n=\infty then U is parabolic and has infinite order, it then fixes the cusp [-1].

EXAMPLES:

```python
sage: from sage.modular.modform_hecketriangle.hecke_triangle_groups import HeckeTriangleGroup
sage: HeckeTriangleGroup(3).U()
[ 1 -1]
[ 1 0]
sage: HeckeTriangleGroup(3).U()^3 == -HeckeTriangleGroup(3).I()
True
sage: HeckeTriangleGroup(3).U()^6 == HeckeTriangleGroup(3).I()
```

(continues on next page)
True
sage: HeckeTriangleGroup(10).U()
[lam -1]
[ 1  0]
True
True
sage: HeckeTriangleGroup(10).U().parent()
Hecke triangle group for n = 10

\textbf{V}(j)

Return the j'th generator for the usual representatives of conjugacy classes of \texttt{self}. It is given by \( V = U^{(j-1)}T \).

**INPUT:**

- \( j \) – Any integer. To get the usual representatives \( j \) should range from 1 to \texttt{self.n()-1}.

**OUTPUT:**

The corresponding matrix/element. The matrix is parabolic if \( j \) is congruent to \(+1\) modulo \texttt{self.n()}. It is elliptic if \( j \) is congruent to 0 modulo \texttt{self.n()}. It is hyperbolic otherwise.

**EXAMPLES:**

```
sage: from sage.modular.modform_hecketriangle.hecke_triangle_groups import ...
    HeckeTriangleGroup
sage: G = HeckeTriangleGroup(3)
sage: G.V(0) == -G.S()
True
sage: G.V(1) == G.T()
True
sage: G.V(2)
[1 0]
[1 1]
sage: G.V(3) == G.S()
True

sage: G = HeckeTriangleGroup(5)
sage: G.element_repr_method("default")
sage: G.V(1)
[ 1 lam]
[ 0  1]
sage: G.V(2)
[lam lam]
[ 1 lam]
sage: G.V(3)
[lam 1]
[lam lam]
sage: G.V(4)
[ 1 0]
```
alpha()
Return the parameter alpha of self. This is the first parameter of the hypergeometric series used in the calculation of the Hauptmodul of self.

EXAMPLES:

```python
sage: from sage.modular.modform_hecketriangle.hecke_triangle_groups import HeckeTriangleGroup
sage: HeckeTriangleGroup(3).alpha()
1/12
sage: HeckeTriangleGroup(4).alpha()
1/8
sage: HeckeTriangleGroup(5).alpha()
3/20
sage: HeckeTriangleGroup(6).alpha()
1/6
sage: HeckeTriangleGroup(10).alpha()
1/5
sage: HeckeTriangleGroup(infinity).alpha()
1/4
```

base_field()
Return the base field of self.

EXAMPLES:

```python
sage: from sage.modular.modform_hecketriangle.hecke_triangle_groups import HeckeTriangleGroup
sage: HeckeTriangleGroup(n=infinity).base_field()
Rational Field
sage: HeckeTriangleGroup(n=7).base_field()
Number Field in lam with defining polynomial x^3 - x^2 - 2*x + 1 with lam = 1.801937735804839?
```

base_ring()
Return the base ring of self.

EXAMPLES:

```python
sage: from sage.modular.modform_hecketriangle.hecke_triangle_groups import HeckeTriangleGroup
sage: HeckeTriangleGroup(n=infinity).base_ring()
Integer Ring
sage: HeckeTriangleGroup(n=7).base_ring()
Maximal Order in Number Field in lam with defining polynomial x^3 - x^2 - 2*x + 1 with lam = 1.801937735804839?
```

beta()
Return the parameter beta of self. This is the second parameter of the hypergeometric series used in the calculation of the Hauptmodul of self.
EXAMPLES:

```python
sage: from sage.modular.modform_hecketriangle.hecke_triangle_groups import HeckeTriangleGroup
sage: HeckeTriangleGroup(3).beta()
5/12
sage: HeckeTriangleGroup(4).beta()
3/8
sage: HeckeTriangleGroup(5).beta()
7/20
sage: HeckeTriangleGroup(6).beta()
1/3
sage: HeckeTriangleGroup(10).beta()
3/10
sage: HeckeTriangleGroup(infinity).beta()
1/4
```

```
class_number(D, primitive=True)
```

Return the class number of self for the discriminant D.

This is the number of conjugacy classes of (primitive) elements of discriminant D.

Note: Due to the 1-1 correspondence with hyperbolic fixed points resp. hyperbolic binary quadratic forms this also gives the class number in those cases.

INPUT:

- D – An element of the base ring corresponding to a valid discriminant.
- primitive – If True (default) then only primitive elements are considered.

EXAMPLES:

```python
sage: from sage.modular.modform_hecketriangle.hecke_triangle_groups import HeckeTriangleGroup
sage: G = HeckeTriangleGroup(n=4)
sage: G.class_number(4)
1
sage: G.class_number(4, primitive=False)
1
sage: G.class_number(14)
2
sage: G.class_number(32)
2
sage: G.class_number(32, primitive=False)
3
sage: G.class_number(68)
4
```

```
class_representatives(D, primitive=True)
```

Return a representative for each conjugacy class for the discriminant D (ignoring the sign).

If primitive=True only one representative for each fixed point is returned (ignoring sign).

INPUT:
• D – An element of the base ring corresponding to a valid discriminant.

• primitive – If True (default) then only primitive representatives are considered.

EXAMPLES:

```python
sage: from sage.modular.modform_hecketriangle.hecke_triangle_groups import HeckeTriangleGroup
sage: G = HeckeTriangleGroup(n=4)
sage: G.element_repr_method("conj")

sage: R = G.class_representatives(4)
sage: R
[[V(2)]]
sage: [v.continued_fraction()[1] for v in R]
[(2,)]

sage: R = G.class_representatives(0)
sage: R
[[V(3)]]
sage: [v.continued_fraction()[1] for v in R]
[(1, 2)]

sage: R = G.class_representatives(-4)
sage: R
[[S]]
sage: R = G.class_representatives(-4, primitive=False)
sage: R
[[S], [U^2]]

sage: R = G.class_representatives(G.lam()^2 - 4)
sage: R
[[U]]
sage: R = G.class_representatives(G.lam()^2 - 4, primitive=False)
sage: R
[[U], [U^(-1)]]

sage: R = G.class_representatives(14)
sage: sorted(R)
[[V(2)*V(3)], [V(1)*V(2)]]
sage: sorted(v.continued_fraction()[1] for v in R)
[(1, 2, 2), (3,)]

sage: R = G.class_representatives(32)
sage: sorted(R)
[[V(3)^2*V(1)], [V(1)^2*V(3)]]
sage: sorted(v.continued_fraction()[1] for v in sorted(R))
[(1, 2, 1, 3), (1, 4)]

sage: R = G.class_representatives(32, primitive=False)
sage: sorted(R)
[[V(3)^2*V(1)], [V(1)^2*V(3)], [V(2)^2]]
```

(continues on next page)
sage: G.element_repr_method("default")

dvalue()

Return a symbolic expression (or an exact value in case \( n=3, 4, 6 \)) for the transfinite diameter (or capacity) of \( \self \).

This is the first nontrivial Fourier coefficient of the Hauptmodul for the Hecke triangle group in case it is normalized to \( \Jinv(i)=1 \).

EXAMPLES:

sage: from sage.modular.modform_hecketriangle.hecke_triangle_groups import HeckeTriangleGroup
sage: HeckeTriangleGroup(3).dvalue()
1/1728
sage: HeckeTriangleGroup(4).dvalue()
1/256
sage: HeckeTriangleGroup(5).dvalue()
e^{(2*euler_gamma - 4*pi/sqrt(5) + 1) + psi(17/20) + psi(13/20))
sage: HeckeTriangleGroup(6).dvalue()
1/108
sage: HeckeTriangleGroup(10).dvalue()
e^{(2*euler_gamma - 4*pi/sqrt(2*sqrt(5) + 10) + psi(4/5) + psi(7/10))}
sage: HeckeTriangleGroup(infinity).dvalue()
1/64

element_repr_method(method=None)

Either return or set the representation method for elements of \( \self \).

INPUT:

- **method** -- If \( \text{method} = \text{None} \) (default) the current default representation method is returned. Otherwise the default method is set to \( \text{method} \). If \( \text{method} \) is not available a \( \text{ValueError} \) is raised. Possible methods are:
  - \( \text{default} \): Use the usual representation method for matrix group elements.
  - \( \text{basic} \): The representation is given as a word in \( S \) and powers of \( T \).
  - \( \text{conj} \): The conjugacy representative of the element is represented as a word in powers of the basic blocks, together with an unspecified conjugation matrix.
  - \( \text{block} \): Same as \( \text{conj} \) but the conjugation matrix is specified as well.

EXAMPLES:

sage: from sage.modular.modform_hecketriangle.hecke_triangle_groups import HeckeTriangleGroup
sage: G = HeckeTriangleGroup(5)
sage: G.element_repr_method()
'default'
sage: G.element_repr_method("basic")
sage: G.element_repr_method()
'basic'
get_FD(z)

Return a tuple (A,w) which determines how to map z to the usual (strict) fundamental domain of self.

INPUT:
• z – a complex number or an element of AlgebraicField().

OUTPUT:
A tuple (A, w).

• A – a matrix in self such that A.acton(w)==z (if z is exact at least).
• w – a complex number or an element of AlgebraicField() (depending on the type z) which lies inside the (strict) fundamental domain of self (self.in_FD(w)==True) and which is equivalent to z (by the above property).

EXAMPLES:

```python
sage: from sage.modular.modform_hecketriangle.hecke_triangle_groups import HeckeTriangleGroup
sage: G = HeckeTriangleGroup(8)
sage: z = AlgebraicField()(1+i/2)
sage: (A, w) = G.get_FD(z)
sage: A
[-lam 1]
[ -1 0]
sage: A.acton(w) == z
True
sage: from sage.modular.modform_hecketriangle.space import ModularForms
sage: z = (134.12 + 0.22*i).n()
sage: (A, w) = G.get_FD(z)
sage: A
[-73*lam^3 + 74*lam 73*lam^2 - 1]
[ -lam^2 + 1 lam]
sage: w
0.769070776942... + 0.779859114103...*I
sage: z
134.12000000... + 0.220000000000...*I
sage: A.acton(w)
134.1200000... + 0.220000000000...*I
```

in_FD(z)

Returns True if z lies in the (strict) fundamental domain of self.

EXAMPLES:

```python
sage: from sage.modular.modform_hecketriangle.hecke_triangle_groups import HeckeTriangleGroup
sage: HeckeTriangleGroup(5).in_FD(CC(1.5/2 + 0.9*i))
True
sage: HeckeTriangleGroup(4).in_FD(CC(1.5/2 + 0.9*i))
False
```

is_arithmetic()

Return True if self is an arithmetic subgroup.
EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.hecke_triangle_groups import HeckeTriangleGroup
sage: HeckeTriangleGroup(3).is_arithmetic()
True
sage: HeckeTriangleGroup(4).is_arithmetic()
True
sage: HeckeTriangleGroup(5).is_arithmetic()
False
sage: HeckeTriangleGroup(6).is_arithmetic()
True
sage: HeckeTriangleGroup(10).is_arithmetic()
False
sage: HeckeTriangleGroup(infinity).is_arithmetic()
True
```

**is_discriminant** \((D, \text{primitive}=\text{True})\)

Returns whether \(D\) is a discriminant of an element of \(\text{self}\).

Note: Checking that something isn’t a discriminant takes much longer than checking for valid discriminants.

**INPUT:**

- \(D\) – An element of the base ring.
- \(\text{primitive}\) – If \(\text{True}\) (default) then only primitive elements are considered.

**OUTPUT:**

True if \(D\) is a primitive discriminant (a discriminant of a primitive element) and False otherwise. If \(\text{primitive=False}\) then also non-primitive elements are considered.

**EXAMPLES:**

```
sage: from sage.modular.modform_hecketriangle.hecke_triangle_groups import HeckeTriangleGroup
sage: G = HeckeTriangleGroup(n=4)
sage: G.is_discriminant(68)
True
sage: G.is_discriminant(196, \text{primitive=\text{False}}) \# long time
True
sage: G.is_discriminant(2)
False
```

**lam()**

Return the parameter lambda of \(\text{self}\), where lambda is twice the real part of rho, lying between 1 (when \(n=3\)) and 2 (when \(n=\text{infinity}\)).

**EXAMPLES:**

```
sage: from sage.modular.modform_hecketriangle.hecke_triangle_groups import HeckeTriangleGroup
sage: HeckeTriangleGroup(3).lam()
1
```

(continues on next page)
sage: HeckeTriangleGroup(4).lam()
lam
sage: HeckeTriangleGroup(4).lam()^2
2
sage: HeckeTriangleGroup(6).lam()^2
3
sage: AA(HeckeTriangleGroup(10).lam())
1.9021130325903...

```sage
sage: lam_minpoly()
Return the minimal polynomial of the corresponding lambda parameter of self.
EXAMPLES:
sage: from sage.modular.modform_hecketriangle.hecke_triangle_groups import HeckeTriangleGroup
sage: HeckeTriangleGroup(10).lam_minpoly()
x^4 - 5*x^2 + 5
sage: HeckeTriangleGroup(17).lam_minpoly()
x^8 - x^7 - 7*x^6 + 6*x^5 + 15*x^4 - 10*x^3 - 10*x^2 + 4*x + 1
sage: HeckeTriangleGroup(infinity).lam_minpoly()
x - 2
```

```sage
sage: list_discriminants(D, primitive=True, hyperbolic=True, incomplete=False)
Returns a list of all discriminants up to some upper bound D.

INPUT:

- **D** – An element/discriminant of the base ring or more generally an upper bound for the discriminant.

- **primitive** – If True (default) then only primitive discriminants are listed.

- **hyperbolic** – If True (default) then only positive discriminants are listed.

- **incomplete** – If True (default: False) then all (also higher) discriminants which were gathered so far are listed (however there might be missing discriminants inbetween).

OUTPUT:

A list of discriminants less than or equal to D.

EXAMPLES:
sage: from sage.modular.modform_hecketriangle.hecke_triangle_groups import HeckeTriangleGroup
sage: G = HeckeTriangleGroup(n=4)
sage: G.list_discriminants(D=68)
[4, 12, 14, 28, 32, 46, 69, 68]
sage: G.list_discriminants(D=0, hyperbolic=False, primitive=False)
[-4, -2, 0]
```
Modular Forms, Release 10.0

sage: G = HeckeTriangleGroup(n=5)
sage: G.list_discriminants(D=20)
[4*lam, 7*lam + 6, 9*lam + 5]
sage: G.list_discriminants(D=0, hyperbolic=False, primitive=False)
[-4, -lam - 2, lam - 3, 0]

n()

Return the parameter n of self, where pi/n is the angle at rho of the corresponding basic hyperbolic triangle with vertices i, rho and infinity.

EXAMPLES:

sage: from sage.modular.modform_hecketriangle.hecke_triangle_groups import HeckeTriangleGroup
sage: HeckeTriangleGroup(10).n()
10
sage: HeckeTriangleGroup(infinity).n()
+Infinity

one()

Return the identity element/matrix for self.

EXAMPLES:

sage: from sage.modular.modform_hecketriangle.hecke_triangle_groups import HeckeTriangleGroup
sage: G = HeckeTriangleGroup(10)
sage: G(1) == G.one()
True
sage: G(1)
[1 0]
[0 1]

rational_period_functions(k, D)

Return a list of basic rational period functions of weight k for discriminant D. The list is expected to be a generating set for all rational period functions of the given weight and discriminant (unknown).

The method assumes that D > 0. Also see the element method rational_period_function for more information.

• k – An even integer, the desired weight of the rational period functions.

• D – An element of the base ring corresponding to a valid discriminant.

EXAMPLES:

sage: from sage.modular.modform_hecketriangle.hecke_triangle_groups import HeckeTriangleGroup
sage: G = HeckeTriangleGroup(n=4)
sage: sorted(G.rational_period_functions(k=4, D=12))
[\((z^4 - 1)/z^4\)]

```
sage: sorted(G.rational_period_functions(k=-2, D=12))
[-z^2 + 1, 4*lam*z^2 - 4*lam]
```

```
sage: sorted(G.rational_period_functions(k=2, D=14))
[(24*z^6 - 120*z^4 + 120*z^2 - 24)/(9*z^8 - 80*z^6 + 146*z^4 - 80*z^2 + 9),
 (24*z^6 - 120*z^4 + 120*z^2 - 24)/(9*z^8 - 80*z^6 + 146*z^4 - 80*z^2 + 9),
 1/z,
 (z^2 - 1)/z^2]
```

```
sage: sorted(G.rational_period_functions(k=-4, D=14))
[-16*z^4 + 16, -z^4 + 1, 16*z^4 - 16]
```

### reduced_elements(D)

Return all reduced (primitive) elements of discriminant \(D\). Also see the element method \(\text{is\_reduced()}\) for more information.

- \(D\) – An element of the base ring corresponding to a valid discriminant.

**EXAMPLES:**

```
sage: from sage.modular.modform hecketriangle.hecke_triangle_groups import HeckeTriangleGroup
sage: G = HeckeTriangleGroup(n=4)
sage: R = G.reduced_elements(D=12)
sage: R
[[5 -lam], [5 -3*lam]
[3*lam -1], [ lam -1]]
sage: [v.continued_fraction() for v in R]
[((), (1, 3)), ((), (3, 1))]
sage: R = G.reduced_elements(D=14)
sage: sorted(R)
[[[3*lam -1] [4*lam -3] [ 5*lam -7] [ 5*lam -3]
[ 1 0], [ 3 -lam], [ 3 -2*lam], [ 7 -2*lam]]
sage: sorted(v.continued_fraction() for v in R)
[(((), (1, 2, 2)), ((), (2, 1, 2)), ((), (2, 2, 1)), ((), (3,)))
```

### rho()

Return the vertex \(\rho\) of the basic hyperbolic triangle which describes \(\text{self}\). \(\rho\) has absolute value 1 and angle \(\pi/n\).

**EXAMPLES:**

```
sage: from sage.modular.modform hecketriangle.hecke_triangle_groups import HeckeTriangleGroup
sage: HeckeTriangleGroup(3).rho() == QQbar(1/2 + sqrt(3)/2*i)
True
sage: HeckeTriangleGroup(4).rho() == QQbar(sqrt(2)/2*(1 + i))
True
sage: HeckeTriangleGroup(6).rho() == QQbar(sqrt(3)/2 + 1/2*i)
```

(continues on next page)
True

```
sage: HeckeTriangleGroup(10).rho()
0.95105651629515...? + 0.30901699437494...?*I
```

```
sage: HeckeTriangleGroup(infinity).rho()
1
```

**root_extension_embedding** (*D, K=None*)

Return the correct embedding from the root extension field of the given discriminant *D* to the field *K*.

Also see the method *root_extension_embedding*(K) of *HeckeTriangleGroupElement* for more examples.

**INPUT:**

- *D* – An element of the base ring of *self*
  - corresponding to a discriminant.
- *K* – A field to which we want the (correct) embedding.
  - If *K=None* (default) then *AlgebraicField()* is used for positive *D* and *AlgebraicRealField()* otherwise.

**OUTPUT:**

The corresponding embedding if it was found. Otherwise a ValueError is raised.

**EXAMPLES:**

```
sage: from sage.modular.modform_hecketriangle.hecke_triangle_groups import HeckeTriangleGroup
sage: G = HeckeTriangleGroup(n=infinity)
sage: G.root_extension_embedding(32)
Ring morphism:
  From: Number Field in e with defining polynomial x^2 - 32
  To: Algebraic Real Field
  Defn: e |--> 5.656854249492...?
sage: G.root_extension_embedding(-4)
Ring morphism:
  From: Number Field in e with defining polynomial x^2 + 4
  To: Algebraic Field
  Defn: e |--> 2*I
sage: G.root_extension_embedding(4)
Coercion map:
  From: Rational Field
  To: Algebraic Real Field

sage: G = HeckeTriangleGroup(n=7)
sage: lam = G.lam()
sage: D = 4*lam^2 + 4*lam - 4
sage: G.root_extension_embedding(D, CC)
Relative number field morphism:
  From: Number Field in e with defining polynomial x^2 - 4*lam^2 - 4*lam + 4
  over its base field
  To: Complex Field with 53 bits of precision
  Defn: e |--> 4.02438434522...
    lam |--> 1.80193773580...
```

(continues on next page)
sage: D = lam^2 - 4
sage: G.root_extension_embedding(D)
Relative number field morphism:
  From: Number Field in e with defining polynomial x^2 - lam^2 + 4 over its base field
  To:   Algebraic Field
  Defn: e |--> 0.?... + 0.867767478235...?*I
        lam |--> 1.801937735804...?

root_extension_field(D)

Return the quadratic extension field of the base field by the square root of the given discriminant D.

INPUT:

• D – An element of the base ring of self corresponding to a discriminant.

OUTPUT:

A relative (at most quadratic) extension to the base field of self in the variable e which corresponds to sqrt(D). If the extension degree is 1 then the base field is returned.

The correct embedding is the positive resp. positive imaginary one. Unfortunately no default embedding can be specified for relative number fields yet.

EXAMPLES:

sage: from sage.modular.modform_hecketriangle.hecke_triangle_groups import HeckeTriangleGroup
sage: G = HeckeTriangleGroup(n=inf)
sage: G.root_extension_field(32)
Number Field in e with defining polynomial x^2 - 32
sage: G.root_extension_field(-4)
Number Field in e with defining polynomial x^2 + 4
sage: G.root_extension_field(4) == G.base_field()
True
sage: G = HeckeTriangleGroup(n=7)
sage: lam = G.lam()
sage: D = 4*lam^2 + 4*lam - 4
sage: G.root_extension_field(D)
Number Field in e with defining polynomial x^2 - 4*lam^2 - 4*lam + 4 over its base field
sage: G.root_extension_field(4) == G.base_field()
True
sage: D = lam^2 - 4
sage: G.root_extension_field(D)
Number Field in e with defining polynomial x^2 - lam^2 + 4 over its base field

simple_elements(D)

Return all simple elements of discriminant D. Also see the element method is_simple() for more information.

• D – An element of the base ring corresponding to a valid discriminant.
EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.hecke_triangle_groups import HeckeTriangleGroup
sage: G = HeckeTriangleGroup(n=4)
sage: sorted(G.simple_elements(D=12))
[[1, 1, lam], [3, lam], [lam, 3], [lam, 1]]
sage: sorted(G.simple_elements(D=14))
[[1, lam, 1], [lam, 3], [2*lam, 1, 2*lam, 3], [3, 2*lam], [1, 2*lam], [3, lam], [1, lam]]
```

2.9 Hecke triangle group elements

AUTHORS:

• Jonas Jermann (2014): initial version

```
class sage.modular.modform_hecketriangle.hecke_triangle_group_element.HeckeTriangleGroupElement(parent, M, check=True, **kwargs):
    Bases: MatrixGroupElement_generic
    Elements of HeckeTriangleGroup.
    a()
    Return the upper left entry of self.
    EXAMPLES:
    sage: from sage.modular.modform_hecketriangle.hecke_triangle_groups import HeckeTriangleGroup
    sage: U = HeckeTriangleGroup(n=7).U()
sage: U.a()
lam
    sage: U.a().parent()
    Maximal Order in Number Field in lam with defining polynomial x^3 - x^2 - 2*x + 1 with lam = 1.801937735804839?
    acton(tau)
    Return the image of tau under the action of self by linear fractional transformations or by conjugation in case tau is an element of the parent of self.
    It is possible to act on points of HyperbolicPlane().
    Note: There is a 1-1 correspondence between hyperbolic fixed points and the corresponding primitive element in the stabilizer. The action in the two cases above is compatible with this correspondence.
    ```
• tau – Either an element of self or any
element to which a linear fractional transformation can be applied in the usual way.

In particular infinity is a possible argument and a possible return value.

As mentioned it is also possible to use points of \texttt{HyperbolicPlane()}. 

\textbf{EXAMPLES:}

```
sage: from sage.modular.modform_hecketriangle.hecke_triangle_groups import HeckeTriangleGroup
sage: G = HeckeTriangleGroup(5)
sage: G.S().acton(SR(1 + i/2))
2/5*I - 4/5
sage: G.S().acton(SR(1 + i/2)).parent()
Symbolic Ring
sage: G.S().acton(QQbar(1 + i/2))
2/5*I - 4/5
sage: G.S().acton(QQbar(1 + i/2)).parent()
Algebraic Field
sage: G.S().acton(i + exp(-2))
-1/(e^(-2) + I)
sage: G.S().acton(i + exp(-2)).parent()
Symbolic Ring
sage: G.T().acton(infinity) == infinity
True
sage: G.U().acton(infinity)
lam
sage: G.V(2).acton(-G.lam()) == infinity
True
sage: G.V(2).acton(G.U()) == G.V(2)*G.U()*G.V(2).inverse()
True
sage: G.V(2).inverse().acton(G.U())
\[
\begin{array}{cc}
0 & -1 \\
1 & \text{lam}
\end{array}
\]
sage: p = HyperbolicPlane().PD().get_point(-I/2+1/8)
sage: G.V(2).acton(p)
Point in PD $(-(47*I + 161)*\sqrt{5} - 47*I - 161)/(145*\sqrt{5} + 94*I + 177) + I)/(1*(-(47*I + 161)*\sqrt{5} - 47*I - 161)/(145*\sqrt{5} + 94*I + 177) + 1)$
sage: bool(G.V(2).acton(p).to_model('UHP').coordinates() == G.V(2).acton(p.to_model('UHP').coordinates()))
True
sage: p = HyperbolicPlane().PD().get_point(I)
sage: G.U().acton(p)
Boundary point in PD $1/2*\sqrt{5} - 2*I + 1)/(-1/2*I*\sqrt{5} - 1/2*I + 1)$
sage: G.U().acton(p).to_model('UHP') == HyperbolicPlane().UHP().get_point(G.lam())
True
sage: G.U().acton(p) == HyperbolicPlane().UHP().get_point(G.lam()).to_model('PD ')
```

(continues on next page)
as_hyperbolic_plane_isometry(model='UHP')

Return self as an isometry of HyperbolicPlane() (in the upper half plane model).

EXAMPLES:

```python
sage: from sage.modular.modform_hecketriangle.hecke_triangle_groups import HeckeTriangleGroup
sage: el = HeckeTriangleGroup(7).V(4)
```

isometry in UHP

```python
[lambda^2 - 1 lambda]
[lambda^2 - 1 lambda^2 - 1]
```

```python
sage: el.as_hyperbolic_plane_isometry().parent()
```

Set of Morphisms from Hyperbolic plane in the Upper Half Plane Model to Hyperbolic plane in the Upper Half Plane Model in Category of hyperbolic models of Hyperbolic plane

```python
sage: el.as_hyperbolic_plane_isometry("KM").parent()
```

Set of Morphisms from Hyperbolic plane in the Klein Disk Model to Hyperbolic plane in the Klein Disk Model in Category of hyperbolic models of Hyperbolic plane

b()

Return the upper right entry of self.

EXAMPLES:

```python
sage: from sage.modular.modform_hecketriangle.hecke_triangle_groups import HeckeTriangleGroup
sage: U = HeckeTriangleGroup(n=7).U()
```

```python
-1
```

```python
sage: U.b().parent()
```

Maximal Order in Number Field in lambda with defining polynomial x^3 - x^2 - 2*x + 1 with lambda = 1.801937735804839

block_decomposition()

Return a tuple (L, R, sgn) such that self = sgn * R.acton(prod(L)) = sgn * R.prod(L)*R.inverse().

In the parabolic and hyperbolic case the tuple entries in L are powers of basic block matrices: V(j) = U^(j-1)*T = self.parent().V(j) for 1 <= j <= n-1. In the elliptic case the tuple entries are either S or U.

This decomposition data is (also) described by _block_decomposition_data().

Warning: The case n=infinity is not verified at all and probably wrong!

EXAMPLES:

```python
sage: from sage.modular.modform_hecketriangle.hecke_triangle_groups import HeckeTriangleGroup
sage: G = HeckeTriangleGroup(n=7)
```

(continues on next page)
sage: G.T().block_decomposition()
((T,), T^(-1), 1)
sage: G.V(2).acton(G.T(-3)).block_decomposition()
((-S*T^(-3)*S,), T, 1)
sage: (-G.V(2)^2).block_decomposition()
((T*S*T^2*S*T,), T*S*T, -1)
sage: el = (-G.V(2)^2*G.V(6)*G.V(3)*G.V(2)*G.V(6)*G.V(3))
sage: el.block_decomposition()
((-S*T^(-1)*S, T*S*T*S*T, T*S*T, -S*T^(-1)*S, T*S*T*S*T, T*S*T, -1)
sage: (G.U()^4*G.S()*G.V(2)).acton(el).block_decomposition()
((T*S*T, -S*T^(-1)*S, T*S*T*S*T, T*S*T, -S*T^(-1)*S, T*S*T*S*T, T*S*T*S*T*S*T^→2*S*T, -1)
sage: (G.V(1)^5*G.V(2)*G.V(3)^3).block_decomposition()
((T*S*T*S*T^2*S*T*S*T^2*S*T*S*T, T^5, T*S*T), T^6*S*T, 1)
sage: G.element_repr_method("default")
sage: (-G.I()).block_decomposition()
(([[1 0] [1 0] [-1 0]
[0 1]], [0 1], [0 -1]
)
sage: G.U().block_decomposition()
([lam -1] [1 0] [1 0]
[1 0],), [0 1], [0 1]
)
sage: (-G.S()).block_decomposition()
([0 -1] [-1 0] [-1 0]
[1 0],), [0 -1], [0 -1]
)
sage: (G.V(2)*G.V(3)).acton(G.U()^6).block_decomposition()
([0 1] [-2*lam^2 - 2*lam + 2 -2*lam^2 - 2*lam + 1] [-1 0]
[-1 lam],), [-2*lam^2 + 1 -2*lam^2 - lam + 2], [0 -1]
)
sage: (G.U()^(-6)).block_decomposition()
([lam -1] [1 0] [-1 0]
[1 0],), [0 1], [0 -1]
)
sage: G = HeckeTriangleGroup(n=8)
sage: (G.U()^4).block_decomposition()
([ lam^2 - 1 -lam^3 + 2*lam] [1 0] [1 0]
[lam^3 - 2*lam -lam^2 + 1],), [0 1], [0 1]
)
sage: (G.U()^(-4)).block_decomposition()
(continues on next page)
Return the block length of self. The block length is given by the number of factors used for the decomposition of the conjugacy representative of self described in primitive_representative(). In particular the block length is invariant under conjugation.

The definition is mostly used for parabolic or hyperbolic elements: In particular it gives a lower bound for the (absolute value of) the trace and the discriminant for primitive hyperbolic elements. Namely \( \text{abs}(\text{trace}) \geq \lambda \times \text{block length} \) and \( \text{discriminant} \geq \text{block length}^2 \times \lambda^2 - 4 \).

Warning: The case \( n=\text{infinity} \) is not verified at all and probably wrong!

**INPUT:**

- **primitive** – If True then the conjugacy representative of the primitive part is used instead, default: False.

**OUTPUT:**

An integer. For hyperbolic elements a non-negative integer. For parabolic elements a negative sign corresponds to taking the inverse. For elliptic elements a (non-trivial) integer with minimal absolute value is chosen. For +- the identity element 0 is returned.

**EXAMPLES:**

```python
sage: from sage.modular.modform_hecketriangle.hecke_triangle_groups import HeckeTriangleGroup
sage: G = HeckeTriangleGroup(n=7)
sage: G.T().block_length()
1
sage: G.V(2).acton(G.T(-3)).block_length()
3
sage: G.V(2).acton(G.T(-3)).block_length(primitive=True)
1
sage: (-G.V(2)).block_length()
1
sage: el = -G.V(2)^3*G.V(6)^2*G.V(3)
sage: t = el.block_length()
sage: D = el.discriminant()
sage: trace = el.trace()
sage: (trace, D, t)
(-124*lam^2 - 103*lam + 68, 65417*lam^2 + 52456*lam - 36300, 6)
sage: abs(AA(trace)) >= AA(G.lam()*t)
True
sage: AA(D) >= AA(t^2 * G.lam() - 4)
True
sage: (el^3).block_length(primitive=True) == t
True
sage: el = (G.U()^4*G.S()*G.V(2)).acton(-G.V(2)^3*G.V(6)^2*G.V(3))
```

(continues on next page)
```python
sage: t = el.block_length()
sage: D = el.discriminant()
sage: trace = el.trace()
sage: (trace, D, t)
(-124*lam^2 - 103*lam + 68, 65417*lam^2 + 52456*lam - 36300, 6)
sage: abs(AA(trace)) >= AA(G.lam()*t)
True
sage: AA(D) >= AA(t^2 * G.lam() - 4)
True
sage: (el^(-2)).block_length(primitive=True) == t
True

sage: el = G.V(1)^5*G.V(2)*G.V(3)^3
sage: t = el.block_length()
sage: D = el.discriminant()
sage: trace = el.trace()
sage: (trace, D, t)
(284*lam^2 + 224*lam - 156, 330768*lam^2 + 265232*lam - 183556, 9)
sage: abs(AA(trace)) >= AA(G.lam()*t)
True
sage: AA(D) >= AA(t^2 * G.lam() - 4)
True
sage: (el^(-1)).block_length(primitive=True) == t
True

sage: (G.V(2)^5*G.V(2)*G.V(3)^3).acton(G.U()^6).block_length()
1
sage: (G.V(2)^5*G.V(2)*G.V(3)^3).acton(G.U()^6).block_length(primitive=True)
1

sage: (-G.I()).block_length()
0
sage: G.U().block_length()
1
sage: (-G.S()).block_length()
1

```

**c()**

Return the lower left entry of self.

**conjugacy_type**(ignore_sign=True, primitive=False)

Return a unique description of the conjugacy class of self (by default only up to a sign).

Warning: The case n=infinity is not verified at all and probably wrong!

**INPUT:**
• **ignore_sign** – If True (default) then the conjugacy classes are only considered up to a sign.

• **primitive** – If True then the conjugacy class of the primitive part is considered instead and the sign is ignored, default: False.

**OUTPUT:**

A unique representative for the given block data (without the conjugation matrix) among all cyclic permutations. If ignore_sign=True then the sign is excluded as well.

**EXAMPLES:**

```python
sage: from sage.modular.modformhecketriangle.hecke_triangle_groups import HeckeTriangleGroup
sage: G = HeckeTriangleGroup(n=7)
sage: (-G.I()).conjugacy_type()
((6, 0),)
sage: G.U().acton(G.S()).conjugacy_type()
((0, 1),
sage: (G.U()^4).conjugacy_type()
(1, -3)
sage: ((G.V(2)*G.V(3)^2*G.V(2)*G.V(3))^2).conjugacy_type()
((3, 2), (2, 1), (3, 1), (2, 1), (3, 2), (2, 1), (3, 1), (2, 1))

sage: (-G.I()).conjugacy_type(ignore_sign=False)
(((6, 0),), -1)
sage: G.S().conjugacy_type(ignore_sign=False)
((0, 1), 1)
sage: (G.U()^4).conjugacy_type(ignore_sign=False)
((1, -3), -1)
sage: G.U().acton((G.V(2)*G.V(3)^2*G.V(2)*G.V(3))^2).conjugacy_type(ignore_sign=False)
(((3, 2), (2, 1), (3, 1), (2, 1), (3, 2), (2, 1), (3, 1), (2, 1)), 1)

sage: (-G.I()).conjugacy_type(primitive=True)
((6, 0),)
sage: G.S().conjugacy_type(primitive=True)
((0, 1),)
sage: G.V(2).acton(G.U()^4).conjugacy_type(primitive=True)
(1, 1)
sage: (G.V(3)^2).conjugacy_type(primitive=True)
((3, 1),)
sage: G.S().acton((G.V(2)*G.V(3)^2*G.V(2)*G.V(3))^2).conjugacy_type(primitive=True)
(((3, 2), (2, 1), (3, 1), (2, 1)),)
```

**continued_fraction()**

For hyperbolic and parabolic elements: Return the (negative) lambda-continued fraction expansion (lambda-CF) of the (attracting) hyperbolic fixed point of self.

Let \( r_j \) in \( \mathbb{Z} \) for \( j \geq 0 \). A finite lambda-CF is defined as: \([r_0; r_1, \ldots, r_k] := (T^{r_0}S^{r_1} \ldots T^{r_k}S)(\infty)\), where \( S \) and \( T \) are the generators of self. An infinite lambda-CF is defined as a corresponding limit value (k->infinity) if it exists.

In this case the lambda-CF of parabolic and hyperbolic fixed points are returned which have an eventually periodic lambda-CF. The parabolic elements are exactly those with a cyclic permutation of the period \([2,\)
1, ..., 1] with \(n-3\) ones.

Warning: The case \(n=\infty\) is not verified at all and probably wrong!

**OUTPUT:**

A tuple \((\text{preperiod}, \text{period})\) with the preperiod and period tuples of the lambda-CF.

**EXAMPLES:**

```python
sage: from sage.modular.modform_hecketriangle.hecke_triangle_groups import HeckeTriangleGroup
sage: G = HeckeTriangleGroup(n=7)
sage: G.T().continued_fraction()
((0, 1), (1, 1, 1, 1, 2))
sage: G.V(2).acton(G.T(-3)).continued_fraction()
(((), (2, 1, 1, 1, 1))
sage: (-G.V(2)).continued_fraction()
((1,), (2,))
sage: (-G.V(2)^3*G.V(6)^2*G.V(3)).continued_fraction()
((1,), (2, 2, 2, 1, 1, 1, 1, 1, 2, 1, 2, 1, 2))
sage: (G.U()^4*G.S()^2).acton(-G.V(2)^3*G.V(6)^2*G.V(3)).continued_fraction()
((1, 1, 1, 2), (2, 2, 2, 2, 1, 1, 1, 2, 1, 1, 1, 2, 1))
sage: (G.V(1)^5*G.V(2)^3*G.V(3)^3).continued_fraction()
((6,), (2, 1, 2, 1, 2, 1, 7))
sage: G = HeckeTriangleGroup(n=8)
sage: G.T().continued_fraction()
((0, 1), (1, 1, 1, 1, 1, 2))
sage: G.V(2).acton(G.T(-3)).continued_fraction()
(((), (2, 1, 1, 1, 1))
sage: (-G.V(2)).continued_fraction()
((1,), (2,))
sage: (-G.V(2)^3*G.V(6)^2*G.V(3)).continued_fraction()
((1,), (2, 2, 2, 1, 1, 1, 1, 1, 2, 1, 2, 1, 2))
sage: (G.U()^4*G.S()^2*G.V(2)).acton(-G.V(2)^3*G.V(6)^2*G.V(3)).continued_fraction()
((1, 1, 1, 2), (2, 2, 2, 2, 1, 1, 1, 2, 1, 1, 1, 2, 1))
sage: (G.V(1)^5*G.V(2)^3*G.V(3)^3).continued_fraction()
((6,), (2, 1, 2, 1, 2, 1, 7))
sage: (G.V(2)^3*G.V(5)*G.V(1)*G.V(6)^2*G.V(4)).continued_fraction()
((1,), (2, 2, 2, 1, 1, 1, 3, 1, 1, 1, 2, 1, 1, 1, 2, 1, 1, 2, 1, 2))
```

\(d()\)

Return the lower right of \(self\).

**EXAMPLES:**

```python
sage: from sage.modular.modform_hecketriangle.hecke_triangle_groups import HeckeTriangleGroup
sage: U = HeckeTriangleGroup(n=7).U()
sage: U.d()
0
```

\(\text{discriminant()}\)
Return the discriminant of \texttt{self} which corresponds to the discriminant of the corresponding quadratic form of \texttt{self}.

\textbf{EXAMPLES:}

\begin{verbatim}
from sage.modular.modform_hecketriangle.hecke_triangle_groups import HeckeTriangleGroup
G = HeckeTriangleGroup(n=7)
G.V(3).discriminant()
4*lam^2 + 4*lam - 4
AA(G.V(3).discriminant())
16.19566935808922?
\end{verbatim}

\textbf{fixed_points(\texttt{embedded=False, order='default'})}

Return a pair of (mutually conjugate) fixed points of \texttt{self} in a possible quadratic extension of the base field.

\textbf{INPUT:}

\begin{itemize}
\item \texttt{embedded} – If \texttt{True} the fixed points are embedded into \texttt{AlgebraicRealField} resp. \texttt{AlgebraicField}. Default: \texttt{False}.
\item \texttt{order} – If order="none" the fixed points are choosen and ordered according to a fixed formula.
\end{itemize}

If order="sign" the fixed points are always ordered according to the sign in front of the square root.

If order="default" (default) then in case the fixed points are hyperbolic they are ordered according to the sign of the trace of \texttt{self} instead, such that the attracting fixed point comes first.

If order="trace" the fixed points are always ordered according to the sign of the trace of \texttt{self}.

If the trace is zero they are ordered by the sign in front of the square root. In particular the fixed_points in this case remain the same for \texttt{-self}.

\textbf{OUTPUT:}

If \texttt{embedded=True} an element of either \texttt{AlgebraicRealField} or \texttt{AlgebraicField} is returned. Otherwise an element of a relative field extension over the base field of (the parent of) \texttt{self} is returned.

Warning: Relative field extensions don't support default embeddings. So the correct embedding (which is the positive resp. imaginary positive one) has to be choosen.

\textbf{EXAMPLES:}

\begin{verbatim}
from sage.modular.modform_hecketriangle.hecke_triangle_groups import HeckeTriangleGroup
G = HeckeTriangleGroup(n=infinity)
(-G.T(-4)).fixed_points()
(+Infinity, +Infinity)
(-G.S()).fixed_points()
(1/2*e, -1/2*e)
p = (-G.S()).fixed_points(embedded=True)[0]
p
(-G.S()).acton(p) == p
True
(-G.V(2)).fixed_points()
\end{verbatim}
Modular Forms, Release 10.0

(continued from previous page)

(1/2*e, -1/2*e)
sage: (-G.V(2)).fixed_points() == G.V(2).fixed_points()
True
sage: p = (-G.V(2)).fixed_points(embedded=True)[1]
sage: p
-1.732050807568878?
sage: (-G.V(2)).acton(p) == p
True

sage: G = HeckeTriangleGroup(n=7)
sage: (-G.S()).fixed_points()
(1/2*e, -1/2*e)
sage: p = (-G.S()).fixed_points(embedded=True)[1]
sage: p
-I
sage: (-G.S()).acton(p) == p
True

sage: (G.U()^4).fixed_points()
((1/2*lam^2 - 1/2*lam - 1/2)*e + 1/2*lam, (-1/2*lam^2 + 1/2*lam + 1/2)*e + 1/
˓→2*lam)
sage: pts = (G.U()^4).fixed_points(order="trace")
sage: (G.U()^4).fixed_points() == [pts[1], pts[0]]
False
sage: (G.U()^4).fixed_points(order="trace") == (-G.U()^4).fixed_points(order="trace")
True
sage: (G.U()^4).fixed_points() == (G.U()^4).fixed_points(order="none")
True
sage: (-G.U()^4).fixed_points() == (G.U()^4).fixed_points()
True
sage: (-G.U()^4).fixed_points(order="none") == pts
True
sage: p = (G.U()^4).fixed_points(embedded=True)[1]
sage: p
0.9009688679024191? - 0.4338837391175581?*I
sage: (G.U()^4).acton(p) == p
True
sage: (-G.V(5)).fixed_points()
((1/2*lam^2 - 1/2*lam - 1/2)*e, (-1/2*lam^2 + 1/2*lam + 1/2)*e)
sage: (-G.V(5)).fixed_points() == G.V(5).fixed_points()
True
sage: p = (-G.V(5)).fixed_points(embedded=True)[0]
sage: p
0.6671145837954892?
sage: (-G.V(5)).acton(p) == p
True

is_elliptic()

Return whether self is an elliptic matrix.

EXAMPLES:
is_hecke_symmetric()  
Return whether the conjugacy class of the primitive part of self, denoted by [gamma] is Hecke - symmetric: i.e. if [gamma] == [gamma^(-1)].

This is equivalent to self.simple_fixed_point_set() being equal with its Hecke - conjugated set (where each fixed point is replaced by the other (Hecke - conjugated) fixed point.

It is also equivalent to [Q] == [-Q] for the corresponding hyperbolic binary quadratic form Q.

The method assumes that self is hyperbolic.

**Warning:** The case n=infinity is not verified at all and probably wrong!

EXAMPLES:

```sage
defining from sage.modular.modform_hecketriangle.hecke_triangle_groups import *HeckeTriangleGroup
sage: G = HeckeTriangleGroup(n=5)
sage: el = G.V(2)
sage: el.is_hecke_symmetric()  
False
sage: (el.simple_fixed_point_set(), el.inverse().simple_fixed_point_set())
(\{1/2*e, (-1/2*lam + 1/2)*e\}, \{-1/2*e, (1/2*lam - 1/2)*e\})

sage: el = G.V(3)*G.V(2)^(-1)*G.V(1)*G.V(6)
sage: el.is_hecke_symmetric()  
False
sage: el.simple_fixed_point_set() == el.inverse().simple_fixed_point_set()
False

sage: el = G.V(2)*G.V(3)
sage: el.is_hecke_symmetric()  
True
sage: sorted(el.simple_fixed_point_set(), key=str)
[(-lam + 3/2)*e + 1/2*lam - 1,  
(-lam + 3/2)*e - 1/2*lam + 1,  
(lam - 3/2)*e + 1/2*lam - 1,  
(lam - 3/2)*e - 1/2*lam + 1]

sage: el.simple_fixed_point_set() == el.inverse().simple_fixed_point_set()
True
```

is_hyperbolic()  
Return whether self is a hyperbolic matrix.

EXAMPLES:
sage: from sage.modular.modform_hecketriangle.hecke_triangle_groups import HeckeTriangleGroup
sage: G = HeckeTriangleGroup(n=7)
sage: [ G.V(k).is_hyperbolic() for k in range(1,8) ]
[False, True, True, True, True, False, False]
sage: G.U().is_hyperbolic()
False

is_identity()

Return whether self is the identity or minus the identity.

EXAMPLES:

sage: from sage.modular.modform_hecketriangle.hecke_triangle_groups import HeckeTriangleGroup
sage: G = HeckeTriangleGroup(n=7)
sage: [ G.V(k).is_identity() for k in range(1,8) ]
[False, False, False, False, False, False, False]
sage: G.U().is_identity()
False

is_parabolic(exclude_one=False)

Return whether self is a parabolic matrix.

If exclude_one is set, then +- the identity element is not considered parabolic.

EXAMPLES:

sage: from sage.modular.modform_hecketriangle.hecke_triangle_groups import HeckeTriangleGroup
sage: G = HeckeTriangleGroup(n=7)
sage: [ G.V(k).is_parabolic() for k in range(1,8) ]
[True, False, False, False, False, True, False]
sage: G.U().is_parabolic()
False
sage: G.V(6).is_parabolic(exclude_one=True)
True
sage: G.V(7).is_parabolic(exclude_one=True)
False

is_primitive()

Returns whether self is primitive. We call an element primitive if (up to a sign and taking inverses) it generates the full stabilizer subgroup of the corresponding fixed point. In the non-elliptic case this means that primitive elements cannot be written as a non-trivial power of another element.

The notion is mostly used for hyperbolic and parabolic elements.

Warning: The case n=Infinity is not verified at all and probably wrong!

EXAMPLES:

sage: from sage.modular.modform_hecketriangle.hecke_triangle_groups import HeckeTriangleGroup
sage: G = HeckeTriangleGroup(n=7)
sage: G.V(2).acton(G.T(-1)).is_primitive()
True

```python
sage: G.T(3).is_primitive()
False
```

```python
sage: (-G.V(2)^2).is_primitive()
False
```

```python
sage: (G.V(1)^5*G.V(2)^4*G.V(3)^3).is_primitive()
True
```

```python
sage: (-G.I()).is_primitive()
True
```

```python
sage: (-G.U()).is_primitive()
True
```

```python
sage: (-G.S()).is_primitive()
True
```

```python
sage: (G.U()^6).is_primitive()
True
```

```python
sage: G = HeckeTriangleGroup(n=8)
sage: (G.U()^2).is_primitive()
False
```

```python
sage: (G.U()^(-4)).is_primitive()
False
```

```python
sage: (G.U()^(-3)).is_primitive()
True
```

**is_reduced**(require_primitive=True, require_hyperbolic=True)

Returns whether `self` is reduced. We call an element reduced if the associated lambda-CF is purely periodic.

I.e. (in the hyperbolic case) if the associated hyperbolic fixed point (resp. the associated hyperbolic binary quadratic form) is reduced.

Note that if `self` is reduced then the element corresponding to the cyclic permutation of the lambda-CF (which is conjugate to the original element) is again reduced. In particular the reduced elements in the conjugacy class of `self` form a finite cycle.

Elliptic elements and +- identity are not considered reduced.

Warning: The case `n=\infty` is not verified at all and probably wrong!

**INPUT:**

- **require_primitive** – If True (default) then non-primitive elements are not considered reduced.

- **require_hyperbolic** – If True (default) then non-hyperbolic elements are not considered reduced.

**EXAMPLES:**

```python
sage: from sage.modular.modform_hecketriangle.hecke_triangle_groups import HeckeTriangleGroup
sage: G = HeckeTriangleGroup(n=8)
sage: G.I().is_reduced(require_hyperbolic=False)
False
```

```python
sage: G.U().reduce().is_reduced(require_hyperbolic=False)
```

(continues on next page)
is_reflection()
Return whether self is the usual reflection on the unit circle.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.hecke_triangle_groups import HeckeTriangleGroup
sage: (-HeckeTriangleGroup(n=7).S()).is_reflection()
True
sage: HeckeTriangleGroup(n=7).U().is_reflection()
False
```

is_simple()
Return whether self is simple. We call an element simple if it is hyperbolic, primitive, has positive sign
and if the associated hyperbolic fixed points satisfy: \( \alpha' < 0 < \alpha \) where \( \alpha \) is the attracting
fixed point for the element.

I.e. if the associated hyperbolic fixed point (resp. the associated hyperbolic binary quadratic form) is
simple.

There are only finitely many simple elements for a given discriminant. They can be used to provide explicit
descriptions of rational period functions.

Warning: The case \( n=\infty \) is not verified at all and probably wrong!

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.hecke_triangle_groups import HeckeTriangleGroup
sage: G = HeckeTriangleGroup(n=5)
sage: el = G.V(2)
sage: el.is_simple()
True
sage: R = el.simple_elements()
```

(continues on next page)
sage: [v.is_simple() for v in R]
[True]
sage: (fp1, fp2) = R[0].fixed_points(embedded=True)
sage: (1.272019649514069?, -1.272019649514069?)
sage: fp2 < 0 < fp1
True

sage: el = G.V(3)^G.V(2)^(-1)*G.V(1)*G.V(6)
sage: el.is_simple()
False
sage: R = el.simple_elements()
sage: [v.is_simple() for v in R]
[True, True]
sage: (fp1, fp2) = R[1].fixed_points(embedded=True)
sage: fp2 < 0 < fp1
True

is_translation(exclude_one=False)

Return whether self is a translation. If exclude_one = True, then the identity map is not considered as a translation.

EXAMPLES:

sage: from sage.modular.modform.hecketriangle.hecke_triangle_groups import HeckeTriangleGroup
sage: (-HeckeTriangleGroup(n=7).T(-4)).is_translation()
True
sage: (-HeckeTriangleGroup(n=7).I()).is_translation()
True
sage: (-HeckeTriangleGroup(n=7).I()).is_translation(exclude_one=True)
False

linking_number()

Let g denote a holomorphic primitive of E2 in the sense: \(\lambda/(2\pi i) \, d/dz \ g = E2\). Let gamma=self and let \(\log((c*z+d) * \text{sgn}(a+d))\) if \(c, a+d > 0\), resp. \(\log((c*z+d)/i*\text{sgn}(c))\) if \(a+d = 0, c!=0\), resp. 0 if \(c=0\). Let \(k=4 * n / (n-2)\), then:

\[ g(\gamma.g.gamma(z) - g(z) - k*M.gamma(z) \] is equal to \(2\pi i / (n-2) * \text{self}.\linking_number(). \)

In particular it is independent of \(z\) and a conjugacy invariant.
If `self` is hyperbolic then in the classical case \( n=3 \) this is the linking number of the closed geodesic (corresponding to `self`) with the trefoil knot.

**EXAMPLES:**

```python
sage: from sage.modular.modform_hecketriangle.hecke_triangle_groups import HeckeTriangleGroup
sage: from sage.modular.modform_hecketriangle.space import QuasiModularForms

sage: def E2_primitive(z, n=3, prec=10, num_prec=53):
    ...:     G = HeckeTriangleGroup(n=n)
    ...:     MF = QuasiModularForms(group=G, k=2, ep=-1)
    ...:     q = MF.get_q(prec=prec)
    ...:     int_series = integrate((MF.E2().q_expansion(prec=prec) - 1) / q)
    ...:     t_const = (2*pi*i/G.lam()).n(num_prec)
    ...:     d = MF.get_d(fix_d=True, d_num_prec=num_prec)
    ...:     q = exp(t_const * z)
    ...:     return t_const*z + sum([(int_series.coefficients()[m]).subs(d=d) * \( q^{\text{int_series.exponents()[m]}} \) for m in range(len(int_series.coefficients()))])

sage: def M(gamma, z, num_prec=53):
    ...:     a = ComplexField(num_prec)(gamma.a())
    ...:     b = ComplexField(num_prec)(gamma.b())
    ...:     c = ComplexField(num_prec)(gamma.c())
    ...:     d = ComplexField(num_prec)(gamma.d())
    ...:     if c == 0:
    ...:         return 0
    ...:     elif a + d == 0:
    ...:         return log(-i.n(num_prec)*(c*z + d)*sign(c))
    ...:     else:
    ...:         return log((c*z+d)*sign(a+d))

sage: def num_linking_number(A, z, n=3, prec=10, num_prec=53):
    ...:     z = z.n(num_prec)
    ...:     k = 4 * n / (n - 2)
    ...:     return (n-2) / (2*pi*i).n(num_prec) * (E2_primitive(A.acton(z), n=n, prec=prec, num_prec=num_prec) - E2Primitive(z, n=n, prec=prec, num_prec=num_prec) - k*M(A, z, num_prec=num_prec))

sage: G = HeckeTriangleGroup(8)
sage: z = i
sage: for A in [G.S(), G.T(), G.U(), G.U()^(G.n()//2), G.U()^(-3)]:
    ...:     print("A=\{\}": A.string_repr("conj"))
    ...:     num_linking_number(A, z, G.n())
    ...:     A.linking_number()
A=[S]:
0.
A=[V(1)]:
6.
A=[U]:
-2.
```

(continues on next page)
A=[U^4]:
0.596987639289... + 0.926018962976...*I

A=[U^(−3)]:
5.40301236071... + 0.926018962976...*I

sage: z = ComplexField(1000)(- 2.3 + 3.1*i)
sage: B = G.I()
sage: for A in [G.S(), G.T(), G.U(), G.U()^(G.n()//2), G.U()^(−3)]:
    ....:     print("A={}: ",format(A.string_rep("conj")))
    ....:     num_linking_number(B.acton(A), z, G.n(), prec=100, num_prec=1000).
    ....:     n(53)

A=[S]:
6.63923483989...e-31 + 2.45195568651...e-30*I

A=[V(1)]:
6.00000000000...

A=[U]:
-2.00000000000... + 2.45195568651...e-30*I

A=[U^4]:
0.00772492873864... + 0.00668936643212...*I

A=[U^(−3)]:
5.99730551444... + 0.000847636355069...*I

A=[S]:
6.63923483989...e-31 + 2.45195568651...e-30*I

A=[V(1)]:
6.00000000000...

A=[U]:
-2.00000000000... + 2.45195568651...e-30*I

A=[U^4]:
0.00772492873864... + 0.00668936643212...*I

A=[U^(−3)]:
5.99730551444... + 0.000847636355069...*I

A=[S]:
-7.96944791339...e-34 - 9.38956758807...e-34*I

A=[V(1)]:
5.99999997397... - 5.96520311160...e-8*I

A=[U]:
-2.00000000000... - 1.33113963568...e-61*I

A=[U^4]:
-2.32704571946...e-6 + 5.91899385948...e-7*I

A=[U^(−3)]:
6.00000032148... - 1.82676936467...e-7*I

(continues on next page)
sage: A = G.V(2)*G.V(3)
sage: B = G.I()
sage: num_linking_number(B.acton(A), z, G.n(), prec=200, num_prec=5000).n(53) ↓
   → # long time
6.00498424588... - 0.00702329345176...*I
sage: A.linking_number()
6

The numerical properties for anything larger are basically too bad to make nice further tests...

\textbf{primitive_part}(method='cf')

Return the primitive part of self. I.e. a group element A with non-negative trace such that self = sign * A^power, where sign = self.sign() is +- the identity (to correct the sign) and power = self.primitive_power().

The primitive part itself is chosen such that it cannot be written as a non-trivial power of another element. It is a generator of the stabilizer of the corresponding (attracting) fixed point.

If self is elliptic then the primitive part is chosen as a conjugate of S or U.

Warning: The case n=infinity is not verified at all and probably wrong!

\textbf{INPUT:}

\begin{itemize}
  \item \textbf{method} – The method used to determine the primitive part (see \texttt{primitive_representative()}, default: “cf”. The parameter is ignored for elliptic elements or +- the identity.
\end{itemize}

The result should not depend on the method.

\textbf{OUTPUT:}

The primitive part as a group element of self.

\textbf{EXAMPLES:}

\begin{verbatim}
sage: from sage.modular.modform_hecketriangle.hecke_triangle_groups import HeckeTriangleGroup
sage: G = HeckeTriangleGroup(n=7)
sage: G.element_repr_method("block")
sage: G.T().primitive_part()
(T^(-1)*S) * (V(6)) * (T^(-1)*S)^(-1)
sage: G.V(2).acton(G.T(-3)).primitive_part()
(T) * (V(6)) * (T)^(-1)
sage: (-G.V(2)).primitive_part()
(T*S*T) * (V(2)) * (T*S*T)^(-1)
sage: (-G.V(2)^3*G.V(6)^2*G.V(3)).primitive_part()
V(2)^3*V(6)^2*V(3)
sage: (-G.U()^4*G.S()*G.V(2)).acton(-G.V(2)^3*G.V(6)^2*G.V(3)).primitive_part()
(T*S*T*S*T*S*T)^(-1)
sage: (T*S*T*S*T*S*T^2*S*T).acton(-G.V(2)^3*G.V(6)^2*G.V(3)).primitive_part()
(V(3)^3*V(1)^5*V(2)) * (T*S*T)^(-1)
sage: (G.V(2)^2*G.V(3)).acton(G.U()^6).primitive_part()
\end{verbatim}
(-T*S*T^2*S*T*S*T) * (U) * (-T*S*T^2*S*T*S*T)^(-1)

sage: (-G.I()).primitive_part()
1

sage: G.U().primitive_part()
U

sage: (-G.S()).primitive_part()
S

sage: el = (G.V(2)*G.V(3)).acton(G.U()^6)

sage: el.primitive_part()
(-T*S*T^2*S*T*S*T) * (U) * (-T*S*T^2*S*T*S*T)^(-1)

sage: el.primitive_part() == el.primitive_part(method="block")
True

sage: G.T().primitive_part()
(T^(-1)*S) * (V(6)) * (T^(-1)*S)^(-1)

sage: G.T().primitive_part(method="block")
(T^(-1)) * (V(1)) * (T^(-1))^(-1)

sage: G.V(2).acton(G.T(-3)).primitive_part() == G.V(2).acton(G.T(-3)).primitive_part(method="block")
True

sage: (-G.V(2)).primitive_part() == (-G.V(2)).primitive_part(method="block")
True

sage: el = -G.V(2)^3*G.V(6)^2*G.V(3)

sage: el.primitive_part() == el.primitive_part(method="block")
True

sage: el = (G.U()^4*G.S()*G.V(2)).acton(-G.V(2)^3*G.V(6)^2*G.V(3))

sage: el.primitive_part() == el.primitive_part(method="block")
True

sage: el=G.V(1)^5*G.V(2)*G.V(3)^3

sage: el.primitive_part() == el.primitive_part(method="block")
True

sage: G.element_repr_method("default")

**primitive_power**(method="cf")

Return the primitive power of self. I.e. an integer power such that self = sign * primitive_part^power, where sign = self.sign() and primitive_part = self.primitive_part(method).

Warning: For the parabolic case the sign depends on the method: The “cf” method may return a negative power but the “block” method never will.

Warning: The case n=infinity is not verified at all and probably wrong!

**INPUT:**

* method – The method used to determine the primitive power (see primitive_representation()), default: “cf”. The parameter is ignored for elliptic elements or +- the identity.

**OUTPUT:**

An integer. For +- the identity element 0 is returned, for parabolic and hyperbolic elements a pos-
itive integer. And for elliptic elements a (non-zero) integer with minimal absolute value such that \( \text{primitive_part}^\text{power} \) still has a positive sign.

EXAMPLES:

```python
sage: from sage.modular.modform_hecketriangle.hecke_triangle_groups import HeckeTriangleGroup
sage: G = HeckeTriangleGroup(n=7)
\[\text{sage: }G.T().primitive_power()\]
-1
\[\text{sage: }G.V(2).acton(G.T(-3)).primitive_power()\]
3
\[\text{sage: }(-G.V(2)^2).primitive_power()\]
2
\[\text{sage: }el = (\text{el} = (-G.V(2)^*G.V(6)^*G.V(3)^*G.V(2)^*G.V(6)^*G.V(3))\]
\[\text{sage: }el.primitive_power()\]
2
\[\text{sage: }el.primitive_power() == G.V(2).acton(G.T(-3)).primitive_power() == G.V(2).acton(G.T(-3)).\]
\[\text{sage: }el.primitive_power(method='block')\]
True
\[\text{sage: }(-G.I()).primitive_power()\]
0
\[\text{sage: }G.U().primitive_power()\]
1
\[\text{sage: }(-G.S()).primitive_power()\]
1
\[\text{sage: }el = (\text{el} = (G.V(2)^*G.V(3)).acton(G.U()^6)\]
\[\text{sage: }el.primitive_power()\]
-1
\[\text{sage: }el.primitive_power() == (-el).primitive_power()\]
True
\[\text{sage: }G = HeckeTriangleGroup(n=8)\]
\[\text{sage: }G.U().primitive_power()\]
4
\[\text{sage: }G.U().acton(G.T(-3)).primitive_power() == G.V(2).acton(G.T(-3)).\]
\[\text{sage: }el.primitive_power(method='block')\]
True
```

\text{primitive representative}(\text{method='block'})

Return a tuple \((P, R)\) which gives the decomposition of the primitive part of \(\text{self}\), namely \(R^\ast P^\ast R\). \text{inverse()} into a specific representative \(P\) and the corresponding conjugation matrix \(R\) (the result depends on the method used).

Together they describe the primitive part of \(\text{self}\). I.e. an element which is equal to \(\text{self}\) up to a sign after taking the appropriate power.

See \_\text{primitive block decomposition data()} for a description about the representative in case the default method block is used. Also see \text{primitive_part()} to construct the primitive part of \(\text{self}\).
Warning: The case \( n=\infty \) is not verified at all and probably wrong!

INPUT:

- **method** – block (default) or cf. The method used to determine \( P \) and \( R \). If \( \text{self} \) is elliptic this parameter is ignored and if \( \text{self} \) is \(-\) the identity then the block method is used.

  With block the decomposition described in \_primitive_block_decomposition_data() is used.

  With cf a reduced representative from the lambda-CF of \( \text{self} \) is used (see \texttt{continued_fraction()}). In that case \( P \) corresponds to the period and \( R \) to the preperiod.

OUTPUT:

A tuple \((P, R)\) of group elements such that \( R*P*R.\text{inverse()} \) is a/the primitive part of \( \text{self} \).

EXAMPLES:

```python
sage: from sage.modular.modform_hecketriangle.hecke_triangle_groups import HeckeTriangleGroup
sage: G = HeckeTriangleGroup(n=7)
sage: G.ordinary_element().primitive_representation(method="cf")
(s*T^(-1)*S*T^(-1)*S*T*S, S*T*S)
sage: G.U().acton(G.T(-3)).primitive_representation(method="cf")
(-T*S*T^(-1)*S*T^(-1), 1)
sage: (-G.V(2)).primitive_representation(method="cf")
(T^2*S, T*S)
sage: (-G.V(2)^3*G.V(6)^2*G.V(3)).primitive_representation(method="cf")
(-T^2*S*T^2*S*T^2*S*T*T^2*S*T*S*T*S*T^2*S, T*S)
sage: (G.U()^6).primitive_representation(method="cf")
(-T^2*S*T^2*S*T^2*S*T^2*S*T*S*T*S*T^2*S, T*S*T*S*T*S*T^2*S)
sage: (G.V(1)^5*G.V(2)*G.V(3)).primitive_representation(method="cf")
(T^2*S*T^2*S*T^2*S*T^2*S*T*S*T*S*T^7*S, T^6*S)
sage: (G.V(2)*G.V(3)).acton(G.U()^6).primitive_representation(method="cf")
```

(continues on next page)
(T*S, -T*S*T^2*S*T*S*T)
sage: (el[0]).is_primitive()
True

sage: G.element_repr_method("block")
sage: el = G.T().primitive_representative()
sage: (el[0]).is_primitive()
True
sage: el = G.V(2).acton(G.T(-3)).primitive_representative()
sage: el
((-S*T^(-1)*S) \times (V(6)) \times (-S*T^(-1)*S)^(-1), (T^(-1)) \times (V(1)) \times (T^(-1))^(-1))
sage: (el[0]).is_primitive()
True
sage: el = (-G.V(2)).primitive_representative()
sage: el
((T*S*T) \times (V(2)) \times (T*S*T)^(-1), (T*S*T) \times (V(2)) \times (T*S*T)^(-1))
sage: (el[0]).is_primitive()
True
sage: el = (-G.V(2)^3*G.V(6)^2*G.V(3)).primitive_representative()
sage: el
(V(2)^3*V(6)^2*V(3), 1)
sage: (el[0]).is_primitive()
True
sage: el = (G.U()^4*G.S()*G.V(2)).acton(-G.V(2)^3*G.V(6)^2*G.V(3)).primitive_representative()
sage: el
(V(2)^3*V(6)^2*V(3), (T*S*T*S*T*S*T) \times (V(2)*V(4)) \times (T*S*T*S*T*S*T)^(-1))
sage: (el[0]).is_primitive()
True
sage: el = (G.V(1)^5*G.V(2)*G.V(3)^3).primitive_representative()
sage: el
(V(3)^3*V(1)^5*V(2), (T^6*S*T) \times (V(1)^5*V(2)) \times (T^6*S*T)^(-1))
sage: (el[0]).is_primitive()
True

sage: G.element_repr_method("default")
sage: el = G.I().primitive_representative()
sage: el
([1 0] [1 0]
[0 1], [0 1]
)
sage: (el[0]).is_primitive()
True

sage: el = G.U().primitive_representative()
sage: el
([lam -1] [1 0]
[ 1 0], [0 1]
)
sage: (el[0]).is_primitive()
rational_period_function$(k)$

The method assumes that $\text{self}$ is hyperbolic.

Return the rational period function of weight $k$ for the primitive conjugacy class of $\text{self}$.

A rational period function of weight $k$ is a rational function $q$ which satisfies: $q + q|S == 0$ and $q + q|U + q|U^2 + \ldots + q|U^{(n-1)} == 0$, where $S = \text{self.parent().S()}$, $U = \text{self.parent().U()}$ and $|$ is the usual slash operator of weight $k$. Note that if $k < 0$ then $q$ is a polynomial.

This method returns a very basic rational period function associated with the primitive conjugacy class of $\text{self}$. The (strong) expectation is that all rational period functions are formed by linear combinations of such functions.

There is also a close relation with modular integrals of weight $2-k$ and sometimes $2-k$ is used for the weight instead of $k$.

Warning: The case $n=\infty$ is not verified at all and probably wrong!

EXAMPLES:

```python
sage: from sage.modular.modformhecketriangle.hecke_triangle_groups import _HeckeTriangleGroup
sage: G = _HeckeTriangleGroup(n=5)
sage: S = G.S()
sage: U = G.U()

sage: def is_rpf(f, k=None):
    ....:     if not f + S.slash(f, k=k) == 0:
    ....:         return False
    ....:     return True

sage: z = PolynomialRing(G.base_ring(), 'z').gen()
sage: [is_rpf(1 - z^(-k), k=k) for k in range(-6, 6, 2)] # long time
[True, True, True, True, True, True]
```

(continues on next page)
Modular Forms, Release 10.0

[False, False, False, False, True, False]

sage: el = G.V(2)
sage: el.is_hecke_symmetric()
False
sage: rpf = el.rational_period_function(-4)
sage: is_rpf(rpf) == is_rpf(rpf, k=-4)
True
sage: is_rpf(rpf)
True
sage: is_rpf(rpf, k=-6)
False
sage: is_rpf(rpf, k=2)
False
sage: rpf
-lam*z^4 + lam
sage: rpf = el.rational_period_function(-2)
sage: is_rpf(rpf)
True
sage: rpf
(lam + 1)*z^2 - lam - 1
sage: el.rational_period_function(0) == 0
True
sage: is_rpf(rpf)
True
sage: rpf
((lam + 1)*z^2 - lam - 1)/((lam*z^4 + (-lam - 2)*z^2 + lam)

sage: el = G.V(3)*G.V(2)^(-1)*G.V(1)*G.V(6)
sage: el.is_hecke_symmetric()
False
sage: rpf = el.rational_period_function(-6)
sage: is_rpf(rpf)
True
sage: rpf
(68*lam + 44)*z^6 + (-24*lam - 12)*z^4 + (24*lam + 12)*z^2 - 68*lam - 44
sage: rpf = el.rational_period_function(-2)
sage: is_rpf(rpf)
True
sage: rpf
((4*lam + 4)*z^2 - 4*lam - 4)

2.9. Hecke triangle group elements

(continues on next page)
sage: el = G.V(2)*G.V(3)

sage: el.is_hecke_symmetric()
True

sage: el.rational_period_function(-4) == 0
True

sage: rpf = el.rational_period_function(-2)

sage: is_rpf(rpf)
True

sage: rpf
(8*lam + 4)*z^2 - 8*lam - 4

sage: el.rational_period_function(0) == 0
True

sage: rpf = el.rational_period_function(2)

sage: is_rpf(rpf)
True

sage: rpf.denominator()
(144*lam + 89)*z^8 + (-618*lam - 382)*z^6 + (951*lam + 588)*z^4 + (-618*lam -
\rightarrow 382)*z^2 + 144*lam + 89

sage: el.rational_period_function(4) == 0
True

\section{reduce(\texttt{primitive=True})}

Return a reduced version of \texttt{self} (with the same the same fixed points). Also see \texttt{is_reduced()}. If \texttt{self} is elliptic (or + the identity) the result is never reduced (by definition). Instead a more canonical conjugation representative of \texttt{self} (resp. it's primitive part) is chosen.

Warning: The case \texttt{n=\textcolor{red}{\texttt{infinity}}} is not verified at all and probably wrong!

INPUT:

\begin{itemize}
  \item \textbf{primitive} – If \texttt{True} (default) then a primitive
    representative for \texttt{self} is returned.
\end{itemize}

EXAMPLES:

sage: from sage.modular.modform hecketriangle hecke_triangle_groups import HeckeTriangleGroup

sage: G = HeckeTriangleGroup(n=7)

sage: print(G.T().reduce().string_repr(\"basic\") + S*T^(-1)*S*T^(-1)*S*T*S

sage: G.T().reduce().is_reduced(require_hyperbolic=False)
True

sage: print(G.V(2).acton(-G.T(-3)).reduce().string_repr("basic") - T*S*T^(-1)*S*T^S

sage: print(G.V(2).acton(-G.T(-3)).reduce(\texttt{primitive=False}).string_repr("basic") + T*S*T^(-3)*S*T^(-1)

sage: print((-G.V(2)).reduce().string_repr("basic") + T^2*S

sage: (-G.V(2)).reduce().is_reduced()
True

sage: print((-G.V(2)^3*G.V(6)^2*G.V(3)).reduce().string_repr("block")
(-S*T^(-1)) * (V(2)^3*V(6)^2*V(3)) * (-S*T^(-1))^(-1)

(continues on next page)
sage: (-G.V^3*G.V(6)^2*G.V(3)).reduce().is_reduced()
True

sage: print((-G.I()).reduce().string_repr("block"))
1
sage: print(G.U().reduce().string_repr("block"))
U
sage: print((-G.S()).reduce().string_repr("block"))
S
sage: print((G.V(2)*G.V(3)).acton(G.U^6).reduce().string_repr("block"))
U
sage: print((G.V(2)*G.V(3)).acton(G.U^6).reduce(primitive=False).string_repr("block"))
-U^(-1)

reduced_elements()

Return the cycle of reduced elements in the (primitive) conjugacy class of self.

I.e. the set (cycle) of all reduced elements which are conjugate to self.primitive_part(). E.g. self.primitive_representative().reduce().

Also see is_reduced(). In particular the result of this method only depends on the (primitive) conjugacy class of self.

The method assumes that self is hyperbolic or parabolic.

Warning: The case n=infinity is not verified at all and probably wrong!

EXAMPLES:

sage: from sage.modular.modform_hecketriangle.hecke_triangle_groups import HeckeTriangleGroup
sage: G = HeckeTriangleGroup(n=5)
sage: G.element_repr_method("basic")

sage: el = G.V(1)
sage: el.continued_fraction()
((0, 1), (1, 1, 2))
sage: R = el.reduced_elements()
sage: R
[T*S*T*S*T^2*S, T*S*T^2*S*T*S, -T*S*T^(-1)*S*T^(-1)]
sage: [v.continued_fraction() for v in R]
[(()), (1, 1, 2)), (()), (1, 2, 1)), (()), (2, 1, 1))]

sage: el = G.V(3)*G.V(2)^(-1)*G.V(1)*G.V(6)
sage: el.continued_fraction()
((1,), (3,))
sage: R = el.reduced_elements()
sage: [v.continued_fraction() for v in R]
[(()), (3,)]

sage: G.element_repr_method("default")

root_extension_embedding(K=None)

Return the correct embedding from the root extension field to K.
INPUT:

- **K** – A field to which we want the (correct) embedding.
  If K=None (default) then AlgebraicField() is used for elliptic elements and AlgebraicRealField() otherwise.

EXAMPLES:

```python
sage: from sage.modular.modform_hecketriangle.hecke_triangle_groups import HeckeTriangleGroup
sage: G = HeckeTriangleGroup(n=infinity)

sage: fp = (-G.S()).fixed_points()[0]
sage: alg_fp = (-G.S()).root_extension_embedding()(fp)
sage: alg_fp
1*I
sage: alg_fp == (-G.S()).fixed_points(embedded=True)[0]
True

sage: fp = (-G.V(2)).fixed_points()[1]
sage: alg_fp = (-G.V(2)).root_extension_embedding()(fp)
sage: alg_fp
-1.732050807568...?

sage: alg_fp == (-G.V(2)).fixed_points(embedded=True)[1]
True

sage: fp = (-G.U()^4).fixed_points()[0]
sage: alg_fp = (-G.U()^4).root_extension_embedding()(fp)
0.9009688679024...? + 0.4338837391175...?*I

sage: alg_fp == (-G.U()^4).fixed_points(embedded=True)[0]
True

sage: (-G.U()^4).root_extension_embedding(CC)(fp)
0.900968867902... + 0.433883739117...*I
Complex Field with 53 bits of precision
```

(continues on next page)
```
sage: fp = (-G.V(5)).fixed_points()[1]
sage: alg_fp = (-G.V(5)).root_extension_embedding()(fp)
sage: alg_fp
-0.6671145837954...

sage: alg_fp == (-G.V(5)).fixed_points(embedded=True)[1]
True
```

**root_extension_field()**

Return a field extension which contains the fixed points of `self`. Namely the root extension field of the parent for the discriminant of `self`. Also see the parent method `root_extension_field(D)` and `root_extension_embedding()` (which provides the correct embedding).

**EXAMPLES:**

```
sage: from sage.modular.modform_hecketriangle.hecke_triangle_groups import HeckeTriangleGroup
sage: G = HeckeTriangleGroup(n=infinity)
sage: G.V(3).discriminant() 32
sage: G.V(3).root_extension_field() == G.root_extension_field(32) True
sage: G.T().root_extension_field() == G.root_extension_field(G.T().discriminant()) == G.base_field() True
sage: (G.S()).root_extension_field() == G.root_extension_field(G.S().discriminant()) True
sage: G = HeckeTriangleGroup(n=7)
sage: D = G.V(3).discriminant()
sage: D
4*lam^2 + 4*lam - 4
sage: G.V(3).root_extension_field() == G.root_extension_field(D) True
sage: G.U().root_extension_field() == G.root_extension_field(G.U().discriminant()) True
sage: G.V(1).root_extension_field() == G.base_field() True
```

**sign()**

Return the sign element/matrix (+- identity) of `self`. The sign is given by the sign of the trace. if the trace is zero it is instead given by the sign of the lower left entry.

**EXAMPLES:**

```
sage: from sage.modular.modform_hecketriangle.hecke_triangle_groups import HeckeTriangleGroup
sage: G = HeckeTriangleGroup(n=7)
sage: (-G.T(-1)).sign()
[-1 0]
[ 0 -1]
sage: G.S().sign()
```

2.9. Hecke triangle group elements
simple_elements()

Return all simple elements in the primitive conjugacy class of self.

I.e. the set of all simple elements which are conjugate to self.primitive_part().

Also see is_simple(). In particular the result of this method only depends on the (primitive) conjugacy class of self.

The method assumes that self is hyperbolic.

Warning: The case n=∞ is not verified at all and probably wrong!

EXAMPLES:

```python
sage: from sage.modular.modform_hecketriangle.hecke_triangle_groups import HeckeTriangleGroup
sage: G = HeckeTriangleGroup(n=5)
sage: el = G.V(2)
sage: el.continued_fraction()
((1,),(2,))
sage: R = el.simple_elements()
sage: R
[
[ lam lam]
[ 1 lam]
]
sage: R[0].is_simple()
True

sage: el = G.V(3)*G.V(2)^(-1)*G.V(1)*G.V(6)
sage: el.continued_fraction()
((1,), (3,))
sage: R = el.simple_elements()
sage: R
```
[ \begin{array}{cc}
2\lambda & 2\lambda + 1 \\
1 & \lambda \\
\end{array} ]
[ \begin{array}{cc}
\lambda & 2\lambda + 1 \\
1 & 2\lambda \\
\end{array} ]

sage: [v.is_simple() for v in R]
[True, True]

sage: el = G.V(1)^2*G.V(2)*G.V(4)
sage: el.discriminant()
135\lambda + 86
sage: R = el.simple_elements()
sage: R
[\begin{array}{ccc}
3\lambda & 3\lambda + 2 \\
8\lambda + 3 & 3\lambda + 2 \\
5\lambda + 2 & 9\lambda + 6 \\
\end{array} ]
[\begin{array}{ccc}
3\lambda + 4 & 6\lambda + 3 \\
\lambda + 2 & \lambda \\
\lambda + 2 & 4\lambda + 1 \\
\end{array} ]
[\begin{array}{ccc}
2\lambda + 1 & 7\lambda + 4 \\
\lambda + 2 & 7\lambda + 2 \\
\end{array} ]

This agrees with the results (p.16) from Culp-Ressler on binary quadratic forms for Hecke triangle groups:

sage: [v.continued_fraction() for v in R]
[((1,), (1, 1, 4, 2)),
((3,), (2, 1, 1, 4)),
((2,), (2, 1, 1, 4)),
((1,), (2, 1, 1, 4))]

simple_fixed_point_set(extended=True)

Return a set of all attracting fixed points in the conjugacy class of the primitive part of self.

If extended=True (default) then also S.acton(alpha) are added for alpha in the set.

This is a so called irreduciblesystemofpoles for rational period functions for the parent group. I.e. the fixed points occur as a irreducible part of the non-zero pole set of some rational period function and all pole sets are given as a union of such irreducible systems of poles.

The method assumes that self is hyperbolic.

Warning: The case n=\infty is not verified at all and probably wrong!

EXAMPLES:

sage: from sage.modular.modform_hecketriangle.hecke_triangle_groups import...
.sage: G = HeckeTriangleGroup(n=5)
.sage: el = G.V(2)
.sage: el.simple_fixed_point_set()
{1/2*e, (-1/2*lambda + 1/2)*e}
.sage: el.simple_fixed_point_set(extended=False)
{1/2*e}
.sage: el = G.V(3)^*G.V(2)^*(-1)^*G.V(1)^*G.V(6)
.sage: el.simple_fixed_point_set()
{(-lambda + 3/2)*e + 1/2*lambda - 1, (-lambda + 3/2)*e - 1/2*lambda + 1, 1/2*e - 1/2*lambda, 1/2*e + 1/2*lambda - 1, 1/2*e + 1/2*lambda + 1, -1/2*e + 1/2*lambda - 1, -1/2*e + 1/2*lambda + 1, -1/2*e - 1/2*lambda, -1/2*e - 1/2*lambda - 1}
\[2^e + 1/2^\text{lam}\]

\[
\text{sage: el.simple_fixed_point_set(extended=False)}
\]
\[
\{1/2^e - 1/2^\text{lam}, 1/2^e + 1/2^\text{lam}\}
\]

\text{\texttt{slash}(f, \text{tau=None, } k=None)}

Return the \texttt{slash} \textit{operator} of weight \(k\) to applied to \(f\), evaluated at \(\tau\). I.e. \((f|_{k}[self])(\tau)\).

**INPUT:**

- \(f\) – A function in \(\tau\) (or an object for which evaluation at \(\text{self.acton}(\tau)\) makes sense.
- \(\tau\) – Where to evaluate the result.
  This should be a valid argument for \(\text{acton()}.\)
  If \(\tau\) is a point of \(\text{HyperbolicPlane()}\) then its coordinates in the upper half plane model are used.
  Default: \(\text{None}\) in which case \(f\) has to be a rational function / polynomial in one variable and the generator of the polynomial ring is used for \(\tau\). That way \texttt{slash} acts on rational functions / polynomials.
- \(k\) – An even integer.
  Default: \(\text{None}\) in which case \(f\) either has to be a rational function / polynomial in one variable (then -degree is used). Or \(f\) needs to have a \textit{weight} attribute which is then used.

**EXAMPLES:**

\[
\text{\texttt{sage: from sage.modular.modform_hecketriangle.hecke_triangle_groups import HeckeTriangleGroup}}
\]
\[
\text{\texttt{sage: from sage.modular.modform_hecketriangle.space import ModularForms}}
\]
\[
\text{\texttt{sage: G = HeckeTriangleGroup(n=5)}}
\]
\[
\text{\texttt{sage: E4 = ModularForms(group=G, k=4, ep=1).E4()}}
\]
\[
\text{\texttt{sage: z = CC(-1/(2*I+30)-1)}}
\]
\[
\text{\texttt{sage: (G.S()).slash(E4, z)}}
\]
\[
32288.0558881\ldots - 118329.856601\ldots*I
\]
\[
\text{\texttt{sage: (G.V(2)*G.V(3)).slash(E4, z)}}
\]
\[
32288.055892\ldots - 118329.856603\ldots*I
\]
\[
\text{\texttt{sage: E4(z)}}
\]
\[
32288.055881\ldots - 118329.856601\ldots*I
\]
\[
\text{\texttt{sage: z = HyperbolicPlane().PD().get_point(CC(-I/2 + 1/8))}}
\]
\[
\text{\texttt{sage: (G.V(2)*G.V(3)).slash(E4, z)}}
\]
\[
-(21624.437\ldots - 12725.035\ldots*I)/(0.610\ldots + 0.324\ldots*I)^4
\]
\[
\text{\texttt{sage: z = PolynomialRing(G.base_ring(), 'z').gen()}}
\]
\[
\text{\texttt{sage: rat = z^2 + 1/(z-G.lam())}}
\]
\[
\text{\texttt{sage: dr = rat.numerator().degree() - rat.denominator().degree()}}
\]
\[
\text{\texttt{sage: G.S().slash(rat) == G.S().slash(rat, tau=None, k=-dr)}}
\]
\[
\text{\texttt{True}}
\]
\[
\text{\texttt{sage: G.S().slash(rat)}}
\]
\[
(z^6 - lam*z^4 - z^3)/(-lam*z^4 - z^3)
\]
\[
\text{\texttt{sage: G.S().slash(rat, k=0)}}
\]
\[(z^4 - \lambda z^2 - z)/(-\lambda z^4 - z^3)\]
\[
\text{sage: } G.S().\text{slash}(\text{rat, k=-4})
\]
\[
(z^8 - \lambda z^6 - z^5)/(-\lambda z^4 - z^3)
\]

**string_repr**(method='default')

Return a string representation of self using the specified method. This method is used to represent self. The default representation method can be set for the parent with self.parent().

**element_repr_method**(method).

**INPUT:**

- **method** – default: Use the usual representation method for matrix group elements.

  - **basic**: The representation is given as a word in S and powers of T.
    
    Note: If S, T are defined accordingly the output can be used/evaluated directly to recover self.

  - **conj**: The conjugacy representative of the element is represented as a word in powers of the basic blocks, together with an unspecified conjugation matrix.

  - **block**: Same as conj but the conjugation matrix is specified as well.
    
    Note: Assuming S, T, U, V are defined accordingly the output can directly be used/evaluated to recover self.

**Warning:** For n=\text{infinity} the methods conj and block are not verified at all and are probably wrong!

**EXAMPLES:**

\[
\text{sage: from sage.modular.modform_hecketriangle.hecke_triangle_groups import HeckeTriangleGroup}
\]
\[
\text{sage: G = HeckeTriangleGroup(n=5)}
\]
\[
\text{sage: el1 = -G.I()}
\]
\[
\text{sage: el2 = G.S() \cdot G.T(3) \cdot G.S() \cdot G.T(-2)}
\]
\[
\text{sage: el3 = G.V(2) \cdot G.V(3) \cdot 2 \cdot G.V(4)^3}
\]
\[
\text{sage: el4 = G.U()^4}
\]
\[
\text{sage: el5 = (G.V(2) \cdot G.T()).acton(-G.S())}
\]
\[
\text{sage: el4.string_repr(method="basic")}
\]
\[
'S*T^(-1)'
\]
\[
\text{sage: G.element_repr_method("default")}
\]
\[
\text{sage: el1}
\]
\[
\begin{bmatrix}
-1 & 0 \\
0 & -1
\end{bmatrix}
\]
\[
\text{sage: el2}
\]
\[
\begin{bmatrix}
 1 & 2*\lambda \\
3*\lambda & -6*\lambda - 7
\end{bmatrix}
\]
\[
\text{sage: el3}
\]
\[
\begin{bmatrix}
34*\lambda + 19 & 5*\lambda + 4 \\
27*\lambda + 18 & 5*\lambda + 2
\end{bmatrix}
\]
\[
\text{sage: el4}
\]
\[
\begin{bmatrix}
0 & -1 \\
1 & -\lambda
\end{bmatrix}
\]
\[
\text{sage: el5}
\]
\[
\begin{bmatrix}
-7*\lambda - 4 & 9*\lambda + 6
\end{bmatrix}
\]
\begin{verbatim}
[-4*lam - 5  7*lam + 4]

sage: G.element_repr_method("basic")

sage: el1

-1

sage: el2
S*T^3*S*T^(-2)

sage: el3
-T*S*T*S*T^(-1)*S*T^(-2)*S*T^(-4)*S

sage: el4
S*T^(-1)

sage: el5
T*S*T^2*S*T^(-2)*S*T^(-1)

sage: G.element_repr_method("conj")

sage: el1

[-1]

sage: el2
[-V(4)^2*V(1)^3]

sage: el3
[V(3)^2*V(4)^3*V(2)]

sage: el4
[-U^(-1)]

sage: el5
[-S]

sage: G.element_repr_method("block")

sage: el1

-1

sage: el2
-(S*T^3) * (V(4)^2*V(1)^3) * (S*T^3)^(-1)

sage: el3
(T*S*T) * (V(3)^2*V(4)^3*V(2)) * (T*S*T)^(-1)

sage: el4
-U^(-1)

sage: el5
-(T*S*T^2) * (S) * (T*S*T^2)^(-1)

sage: G.element_repr_method("default")

sage: G = HeckeTriangleGroup(n=infinity)

sage: el = G.S()*G.T(3)*G.S()*G.T(-2)

sage: print(el.string_repr())

[-1  4]

[-6 -25]

sage: print(el.string_repr(method="basic"))

S*T^3*S*T^(-2)
\end{verbatim}

trace()

Return the trace of self, which is the sum of the diagonal entries.

EXAMPLES:
word_S_T()

Decompose self into a product of the generators $S$ and $T$ of its parent, together with a sign correction matrix, namely: $self = sgn * \text{prod}(L)$.

Warning: If $self$ is $\pm$ the identity $\text{prod}(L)$ is an empty product which produces 1 instead of the identity matrix.

OUTPUT:

The function returns a tuple $(L, sgn)$ where the entries of $L$ are either the generator $S$ or a non-trivial integer power of the generator $T$. $sgn$ is $\pm$ the identity.

If this decomposition is not possible a TypeError is raised.

EXAMPLES:

```python
sage: from sage.modular.modform_hecketriangle.hecke_triangle_groups import HeckeTriangleGroup
sage: G = HeckeTriangleGroup(n=7)
sage: G.U().trace()
lam
sage: G.S().trace()
0
```

```python
sage: from sage.modular.modform_hecketriangle.hecke_triangle_groups import HeckeTriangleGroup
sage: G = HeckeTriangleGroup(n=17)
sage: (-G.I()).word_S_T()[0]
()
sage: (-G.I()).word_S_T()[1]
[-1 0]
[ 0 -1]
sage: (L, sgn) = (-G.V(2)).word_S_T()
sage: L
([ 1 lam] [ 0 -1] [ 1 lam] [ 0 1], [ 1 0], [ 0 1])
sage: sgn == -G.I()
True
sage: -G.V(2) == sgn * prod(L)
True
sage: (L, sgn) = G.U().word_S_T()
sage: L
([ 1 lam] [ 0 -1] [ 0 1], [ 1 0])
sage: sgn == G.I()
True
sage: G.U() == sgn * prod(L)
True
sage: G = HeckeTriangleGroup(n=infinity)
sage: (L, sgn) = (-G.V(2)*G.V(3)).word_S_T()
```

(continues on next page)
sage: L
( [1 2] [0 -1] [1 4] [0 -1] [1 2] [0 -1] [1 2]
[0 1], [1 0], [0 1], [1 0], [0 1], [1 0], [0 1]
)
sage: -G.V(2)*G.V(3) == sgn * prod(L)
True

sage.modular.modform.hecketriangle.hecke_triangle_group_element.coerce_AA(p)

Return the argument first coerced into AA and then simplified.

This leads to a major performance gain with some operations.

EXAMPLES:

sage: from sage.modular.modform.hecketriangle.hecke_triangle_group_element import...
    → coerce_AA
sage: p = (791264*AA(2*cos(pi/8))^2 - 463492).sqrt()
sage: AA(p)._exact_field()
Number Field in a with defining polynomial y^8 - 1910*y^2 - 3924*y + 681058 with a...
    → in ...

sage.modular.modform.hecketriangle.hecke_triangle_group_element.cyclic_representative(L)

Return a unique representative among all cyclic permutations of the given list/tuple.

INPUT:

• L – A list or tuple.

OUTPUT:

The maximal element among all cyclic permutations with respect to lexicographical ordering.

EXAMPLES:

sage: from sage.modular.modform.hecketriangle.hecke_triangle_group_element import...
    → cyclic_representation
sage: cyclic_representation((1,))
(1,)
sage: cyclic_representation((2,))
(2,)
sage: cyclic_representation((1,2,1,2))
(2, 1, 2, 1)
sage: cyclic_representation((1,2,3,2,3,1))
(3, 2, 3, 1, 1, 2)
2.10 Analytic types of modular forms

Properties of modular forms and their generalizations are assembled into one partially ordered set. See AnalyticType for a list of handled properties.

AUTHORS:
- Jonas Jermann (2013): initial version

class sage.modular.modform_hecketriangle.analytic_type.AnalyticType
Bases: FiniteLatticePoset
Container for all possible analytic types of forms and/or spaces.

The analytic type of forms spaces or rings describes all possible occurring basic analytic properties of elements in the space/ring (or more).

For ambient spaces/rings this means that all elements with those properties (and the restrictions of the space/ring) are contained in the space/ring.

The analytic type of an element is the analytic type of its minimal ambient space/ring.

The basic analytic properties are:
- quasi - Whether the element is quasi modular (and not modular) or modular.
- mero - meromorphic: If the element is meromorphic and meromorphic at infinity.
- weak - weakly holomorphic: If the element is holomorphic and meromorphic at infinity.
- holo - holomorphic: If the element is holomorphic and holomorphic at infinity.
- cusp - cuspidal: If the element additionally has a positive order at infinity.

The zero elements/property have no analytic properties (or only quasi).

For ring elements the property describes whether one of its homogeneous components satisfies that property and the “union” of those properties is returned as the analytic type.

Similarly for quasi forms the property describes whether one of its quasi components satisfies that property.

There is a (natural) partial order between the basic properties (and analytic types) given by “inclusion”. We name the analytic type according to its maximal analytic properties.

For \( n = 3 \) the quasi form \( \text{el} = E_6 - E_2^3 \) has the quasi components \( E_6 \) which is holomorphic and \( E_2^3 \) which is quasi holomorphic. So the analytic type of \( \text{el} \) is quasi holomorphic despite the fact that the sum \( (\text{el}) \) describes a function which is zero at infinity.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.space import QuasiModularForms
dsage: x,y,z,d = var("x,y,z,d")
dsage: el = QuasiModularForms(n=3, k=6, ep=-1)(y-z^3)
dsage: el.analytic_type()
quasi modular
```

Similarly the type of the ring element \( \text{el2} = E_4/\Delta - E_6/\Delta \) is weakly holomorphic despite the fact that the sum \( (\text{el2}) \) describes a function which is holomorphic at infinity.
sage: from sage.modular.modform_hecketriangle.graded_ring import WeakModularFormsRing
sage: x,y,z,d = var("x,y,z,d")

sage: el2 = WeakModularFormsRing(n=3)(x/(x^3-y^2)-y/(x^3-y^2))
sage: el2.analytic_type()

weakly holomorphic modular

Element

alias of AnalyticTypeElement

base_poset()

Return the base poset from which everything of self was constructed. Elements of the base poset correspond to the basic analytic properties.

EXAMPLES:

sage: from sage.modular.modform_hecketriangle.analytic_type import AnalyticType
sage: from sage.combinat.posets.posets import FinitePoset
sage: AT = AnalyticType()
sage: P = AT.base_poset()
sage: P

Finite poset containing 5 elements with distinguished linear extension
sage: isinstance(P, FinitePoset)
True

sage: P.is_lattice()
False
sage: P.is_finite()
True
sage: P.cardinality()
5
sage: P.is_bounded()
False
sage: P.list()
[cusp, holo, weak, mero, quasi]

sage: len(P.relations())
11
sage: P.cover_relations()
[[cusp, holo], [holo, weak], [weak, mero]]

sage: P.has_top()
False
sage: P.has_bottom()
False

lattice_poset()

Return the underlying lattice poset of self.

EXAMPLES:

sage: from sage.modular.modform_hecketriangle.analytic_type import AnalyticType
sage: AnalyticType().lattice_poset()

Finite lattice containing 10 elements
class sage.modular.modform_hecketriangle.analytic_type.AnalyticTypeElement(poset, element, vertex)

Bases: LatticePosetElement

Analytic types of forms and/or spaces.

An analytic type element describes what basic analytic properties are contained/included in it.

EXAMPLES:

```python
sage: from sage.modular.modform_hecketriangle.analytic_type import AnalyticType
sage: from sage.combinat.posets.elements import LatticePosetElement
sage: AT = AnalyticType()
sage: el = AT("quasi", "cusp")
sage: el
quasi cuspidal
sage: isinstance(el, AnalyticTypeElement)
True
sage: isinstance(el, LatticePosetElement)
True
sage: el.parent() == AT
True
sage: sorted(el.element,key=str)
['cusp', 'quasi']
```

```python
sage: from sage.sets.set import Set_object_enumerated
sage: isinstance(el.element, Set_object_enumerated)
True
sage: first = sorted(el.element,key=str)[0]; first
'spseudocusp'
```

```python
sage: el.parent() == AT.base_poset()
True
```

```python
sage: el2 = AT("holo")
sage: sum = el + el2
sage: sum
quasi modular
```

```python
sage: sorted(sum.element,key=str)
['cusp', 'holo', 'quasi']
```

```python
sage: el * el2
cuspidal
```

analytic_name()

Return a string representation of the analytic type.

EXAMPLES:

```python
sage: from sage.modular.modform_hecketriangle.analytic_type import AnalyticType
sage: AT = AnalyticType()
sage: AT(["quasi", "weak"]).analytic_name()
'quasi weakly holomorphic modular'
sage: AT(["quasi", "cusp"]).analytic_name()
'quasi cuspidal'
sage: AT(["quasi"]).analytic_name()
'zero'
```

(continues on next page)
sage: AT([]).analytic_name() 'zero'

analytic_space_name()

Return the (analytic part of the) name of a space with the analytic type of self.

This is used for the string representation of such spaces.

EXAMPLES:

sage: from sage.modular.modform_hecketriangle.analytic_type import AnalyticType
sage: AT = AnalyticType()
sage: AT(["quasi", "weak"]).analytic_space_name() 'QuasiWeakModular'
sage: AT(["quasi", "cusp"]).analytic_space_name() 'QuasiCusp'
sage: AT(["quasi"]).analytic_space_name() 'Zero'
sage: AT([]).analytic_space_name() 'Zero'

extend_by(extend_type)

Return a new analytic type which contains all analytic properties specified either in self or in extend_type.

INPUT:

• extend_type – an analytic type or something which is convertible to an analytic type

OUTPUT:

The new extended analytic type.

EXAMPLES:

sage: from sage.modular.modform_hecketriangle.analytic_type import AnalyticType
sage: AT = AnalyticType()
sage: el = AT(["quasi", "cusp"])
sage: el2 = AT("holo")

sage: el.extend_by(el2)

quasi modular

sage: el.extend_by(el2) == el + el2

True

latex_space_name()

Return the short (analytic part of the) name of a space with the analytic type of self for usage with latex.

This is used for the latex representation of such spaces.

EXAMPLES:

sage: from sage.modular.modform_hecketriangle.analytic_type import AnalyticType
sage: AT = AnalyticType()
sage: AT("mero").latex_space_name() '\tilde{M}'
reduce_to(reduce_type)

Return a new analytic type which contains only analytic properties specified in both self and reduce_type.

INPUT:

• reduce_type – an analytic type or something which is convertible to an analytic type

OUTPUT:

The new reduced analytic type.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.analytic_type import AnalyticType
sage: AT = AnalyticType()
sage: el = AT(["quasi", "cusp"])
sage: el2 = AT("holo")

sage: el.reduce_to(el2)
cuspidal
sage: el.reduce_to(el2) == el * el2
True
```
class sage.modular.modform_hecketriangle.graded_ring.QuasiCuspFormsRing(group, base_ring, red_hom, n)

Bases: FormsRing_abstract, CommutativeAlgebra, UniqueRepresentation

Graded ring of (Hecke) quasi cusp forms for the given group and base ring.

class sage.modular.modform_hecketriangle.graded_ring.QuasiMeromorphicModularFormsRing(group, base_ring, red_hom, n)

Bases: FormsRing_abstract, CommutativeAlgebra, UniqueRepresentation

Graded ring of (Hecke) quasi meromorphic modular forms for the given group and base ring.

class sage.modular.modform_hecketriangle.graded_ring.QuasiModularFormsRing(group, base_ring, red_hom, n)

Bases: FormsRing_abstract, CommutativeAlgebra, UniqueRepresentation

Graded ring of (Hecke) quasi modular forms for the given group and base ring

class sage.modular.modform_hecketriangle.graded_ring.QuasiWeakModularFormsRing(group, base_ring, red_hom, n)

Bases: FormsRing_abstract, CommutativeAlgebra, UniqueRepresentation

Graded ring of (Hecke) quasi weakly holomorphic modular forms for the given group and base ring.

class sage.modular.modform_hecketriangle.graded_ring.WeakModularFormsRing(group, base_ring, red_hom, n)

Bases: FormsRing_abstract, CommutativeAlgebra, UniqueRepresentation

Graded ring of (Hecke) weakly holomorphic modular forms for the given group and base ring

sage.modular.modform_hecketriangle.graded_ring.canonical_parameters(group, base_ring, red_hom, n=None)

Return a canonical version of the parameters.

EXAMPLES:

```python
sage: from sage.modular.modform_hecketriangle.graded_ring import canonical_parameters
sage: canonical_parameters(4, ZZ, 1)
(Hecke triangle group for n = 4, Integer Ring, True, 4)
sage: canonical_parameters(infinity, RR, 0)
(Hecke triangle group for n = +Infinity, Real Field with 53 bits of precision, False, +Infinity)
```

### 2.12 Modular forms for Hecke triangle groups

**AUTHORS:**

- Jonas Jermann (2013): initial version

class sage.modular.modform_hecketriangle.space.CuspForms(group, base_ring, k, ep, n)

Bases: FormsSpace_abstract, Module, UniqueRepresentation

Module of (Hecke) cusp forms for the given group, base ring, weight and multiplier
coordinate_vector(v)
Return the coordinate vector of v with respect to the basis self.gens().

INPUT:
• v – An element of self.

OUTPUT:
An element of self.module(), namely the corresponding coordinate vector of v with respect to the basis self.gens().
The module is the free module over the coefficient ring of self with the dimension of self.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.space import CuspForms
sage: MF = CuspForms(n=12, k=72/5, ep=-1)
sage: MF.default_prec(4)
sage: MF.dimension()
2
sage: el = MF(MF.f_i()*MF.Delta())
sage: el
q - 1/(288*d)*q^2 - 96605/(1327104*d^2)*q^3 + O(q^4)
sage: vec = el.coordinate_vector()
sage: vec
(1, -1/(288*d))
sage: vec.parent()
Vector space of dimension 2 over Fraction Field of Univariate Polynomial Ring in d over Integer Ring
sage: vec.parent() == MF.module()
True
sage: el == vec[0]*MF.gen(0) + vec[1]*MF.gen(1)
True
sage: el == MF.element_from_coordinates(vec)
True
sage: MF = CuspForms(n=infinity, k=16)
sage: el2 = MF(MF.Delta()*MF.E4())
sage: vec2 = el2.coordinate_vector()
sage: vec2
(1, 5/(8*d), 187/(1024*d^2))
sage: el2 == MF.element_from_coordinates(vec2)
True
```

dimension()
Return the dimension of self.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.space import CuspForms
sage: MF = CuspForms(n=infinity, k=16)
sage: el2 = MF(MF.Delta()*MF.E4())
sage: vec2 = el2.coordinate_vector()
sage: vec2
(1, 5/(8*d), 187/(1024*d^2))
sage: el2 == MF.element_from_coordinates(vec2)
True
```
sage: CuspForms(n=infinity, k=8).dimension()
1

gens()

Return a basis of self as a list of basis elements.

EXAMPLES:

sage: from sage.modular.modform_hecketriangle.space import CuspForms
sage: MF=CuspForms(n=12, k=72/5, ep=1)
sage: MF
CuspForms(n=12, k=72/5, ep=1) over Integer Ring
sage: MF.dimension()
3
sage: MF.gens()
[q + 296888795/(10319560704*d^3)*q^4 + O(q^5),
q^2 + 6629/(221184*d^2)*q^4 + O(q^5),
q^3 - 25/(96*d)*q^4 + O(q^5)]

sage: MF = CuspForms(n=infinity, k=8, ep=1)
sage: MF.gen(0) == MF.E4()*MF.f_inf()
True

class sage.modular.modform_hecketriangle.space.MeromorphicModularForms(group, base_ring, k, ep, n)

Bases: FormsSpace_abstract, Module, UniqueRepresentation

Module of (Hecke) meromorphic modular forms for the given group, base ring, weight and multiplier

class sage.modular.modform_hecketriangle.space.ModularForms(group, base_ring, k, ep, n)

Bases: FormsSpace_abstract, Module, UniqueRepresentation

Module of (Hecke) modular forms for the given group, base ring, weight and multiplier

coordinate_vector(v)

Return the coordinate vector of v with respect to the basis self.gens().

INPUT:

• v – An element of self.

OUTPUT:

An element of self.module(), namely the corresponding coordinate vector of v with respect to the basis self.gens().

The module is the free module over the coefficient ring of self with the dimension of self.

EXAMPLES:

sage: from sage.modular.modform_hecketriangle.space import ModularForms
sage: MF = ModularForms(n=6, k=20, ep=1)
sage: MF.dimension()
4
sage: el = MF.E4()^2*MF.Delta()
sage: el
q + 78*q^2 + 2781*q^3 + 59812*q^4 + O(q^5)
sage: vec = el.coordinate_vector()
sage: vec
(0, 1, 13/(18*d), 103/(432*d^2))
sage: vec.parent()
Vector space of dimension 4 over Fraction Field of Univariate Polynomial Ring in d over Integer Ring
sage: vec.parent() == MF.module()
True
sage: el == vec[0]*MF.gen(0) + vec[1]*MF.gen(1) + vec[2]*MF.gen(2) + vec[3]*MF.gen(3)
True
sage: el == MF.element_from_coordinates(vec)
True
sage: MF = ModularForms(n=infinity, k=8, ep=1)
sage: (MF.E4()^2).coordinate_vector()
(1, 1/(2*d), 15/(128*d^2))

**dimension()**
Return the dimension of self.

**EXAMPLES:**

```
sage: from sage.modular.modform_hecketriangle.space import ModularForms
sage: MF = ModularForms(n=6, k=20, ep=1)
sage: MF.dimension()
4
sage: len(MF.gens()) == MF.dimension()
True
sage: ModularForms(n=infinity, k=8).dimension()
3
```

gens()
Return a basis of self as a list of basis elements.

**EXAMPLES:**

```
sage: from sage.modular.modform_hecketriangle.space import ModularForms
sage: MF = ModularForms(n=6, k=20, ep=1)
sage: MF.dimension()
4
sage: MF.gens()
[1 + 360360*q^4 + O(q^5),
 q + 21742*q^4 + O(q^5),
 q^2 + 702*q^4 + O(q^5),
 q^3 - 6*q^4 + O(q^5)]
```

class sage.modular.modform_hecketriangle.space.QuasiCuspForms(group, base_ring, k, ep, n)
Bases: FormsSpace_abstract, Module, UniqueRepresentation
Module of (Hecke) quasi cusp forms for the given group, base ring, weight and multiplier

**coordinate_vector**(*v*)

Return the coordinate vector of *v* with respect to the basis `self.gens()`.

**INPUT:**

- *v* – An element of `self`.

**OUTPUT:**

An element of `self.module()`, namely the corresponding coordinate vector of *v* with respect to the basis `self.gens()`.

The module is the free module over the coefficient ring of `self` with the dimension of `self`.

**EXAMPLES:**

```python
sage: from sage.modular.modform_hecketriangle.space import QuasiCuspForms
sage: MF = QuasiCuspForms(n=6, k=20, ep=1)
sage: MF.dimension()
12
sage: el = MF(MF.E4()^2*MF.Delta() + MF.E4()*MF.E2()^2*MF.Delta())
sage: el
2*q + 120*q^2 + 3402*q^3 + 61520*q^4 + O(q^5)
sage: vec = el.coordinate_vector()  # long time
sage: vec  # long time
(1, 13/(18*d), 103/(432*d^2), 0, 0, 1, 1/(2*d), 0, 0, 0, 0, 0)
sage: vec.parent()  # long time
Vector space of dimension 12 over Fraction Field of Univariate Polynomial Ring in d over Integer Ring
sage: vec.parent() == MF.module()  # long time
True
sage: el == MF(sum([vec[l]*MF.gen(l) for l in range(0,12)]))  # long time
True
sage: el == MF.element_from_coordinates(vec)  # long time
True
sage: MF.gen(1).coordinate_vector() == vector([0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])  # long time
True
sage: MF = QuasiCuspForms(n=infinity, k=10, ep=-1)
sage: el2 = MF(MF.E4()*MF.f_inf()*(MF.f_i() - MF.E2()))
sage: el2.coordinate_vector()
(1, -1)
sage: el2 == MF.element_from_coordinates(el2.coordinate_vector())
True
```

**dimension()**

Return the dimension of `self`.

**EXAMPLES:**

```python
sage: from sage.modular.modform_hecketriangle.space import QuasiCuspForms
sage: MF = QuasiCuspForms(n=8, k=46/3, ep=-1)
sage: MF.default_prec(3)
sage: MF.dimension()
```
sage: len(MF.gens()) == MF.dimension()
True

sage: QuasiCuspForms(n=infinity, k=10, ep=-1).dimension()
2

gens()

Return a basis of self as a list of basis elements.

EXAMPLES:

sage: from sage.modular.modform_hecketriangle.space import QuasiCuspForms
sage: MF = QuasiCuspForms(n=8, k=46/3, ep=-1)

sage: MF.default_prec(4)

sage: MF.gens()
[q - 17535/(262144*d^2)*q^3 + O(q^4),
q^2 - 47/(128*d)*q^3 + O(q^4),
q - 9/(128*d)*q^2 + 15633/(262144*d^2)*q^4 + 0(q^4),
q^2 - 7/(128*d)*q^4 + 0(q^4),
q - 23/(64*d)*q^2 - 3103/(262144*d^2)*q^3 + 0(q^4),
q - 3/(64*d)*q^2 - 4863/(262144*d^2)*q^4 + 0(q^4),
q - 27/(64*d)*q^2 + 17217/(262144*d^2)*q^3 + 0(q^4)]

sage: MF = QuasiCuspForms(n=infinity, k=10, ep=-1)

sage: MF.gens()
[q - 16*q^2 - 156*q^3 - 256*q^4 + O(q^5), q - 60*q^3 - 256*q^4 + O(q^5)]
sage: from sage.modular.modform_hecketriangle.space import QuasiModularForms
sage: MF = QuasiModularForms(n=6, k=20, ep=1)

sage: MF.dimension()
22

sage: el = MF(MF.E4()^2*MF.E6()^2 + MF.E4()^3*MF.E2()^2*MF.Delta() + MF.E2()^3*MF.E4()^2*MF.E6())

sage: el
2 + 25*q - 2478*q^2 - 82731*q^3 - 448484*q^4 + O(q^5)

sage: vec = el.coordinate_vector()  # long time

sage: vec
(1, 1/(9*d), -11/(81*d^2), -4499/(104976*d^3), 0, 0, 0, 0, 1, 1/(2*d), 1, 5/(18*d), 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)

sage: vec.parent()  # long time
Vector space of dimension 22 over Fraction Field of Univariate Polynomial Ring in d over Integer Ring

sage: vec.parent() == MF.module()  # long time
True

sage: el == MF(sum([vec[l]*MF.gen(l) for l in range(0,22)]))  # long time
True

sage: el == MF.element_from_coordinates(vec)  # long time
True

sage: MF.gen(1).coordinate_vector() == vector([0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])  # long time
True

sage: MF = QuasiModularForms(n=infinity, k=4, ep=1)

sage: el2 = MF.E4() + MF.E2()^2

sage: el2
2 + 160*q^2 + 512*q^3 + 1632*q^4 + O(q^5)

sage: el2.coordinate_vector()
(1, 1/(4*d), 0, 1)

sage: el2 == MF.element_from_coordinates(el2.coordinate_vector())
True

---

dimension()

Return the dimension of self.

EXAMPLES:

sage: from sage.modular.modform_hecketriangle.space import QuasiModularForms
sage: MF = QuasiModularForms(n=5, k=6, ep=-1)

sage: MF.dimension()
3

sage: len(MF.gens()) == MF.dimension()
True

 gens()

Return a basis of self as a list of basis elements.

EXAMPLES:

sage: from sage.modular.modform_hecketriangle.space import QuasiModularForms
sage: MF = QuasiModularForms(n=5, k=6, ep=-1)

sage: MF.default_prec(2)

(continues on next page)
sage: MF.gens()
[1 - 37/(200*d)*q + O(q^2),
 1 + 33/(200*d)*q + O(q^2),
 1 - 27/(200*d)*q + O(q^2)]

sage: MF = QuasiModularForms(n=infinity, k=2, ep=-1)
sage: MF.default_prec(2)
sage: MF.gens()
[1 - 24*q + O(q^2), 1 - 8*q + O(q^2)]

class sage.modular.modform_hecketriangle.space.QuasiWeakModularForms(group, base_ring, k, ep, n)
Bases: FormsSpace_abstract, Module, UniqueRepresentation

Module of (Hecke) quasi weakly holomorphic modular forms for the given group, base ring, weight and multiplier

class sage.modular.modform_hecketriangle.space.WeakModularForms(group, base_ring, k, ep, n)
Bases: FormsSpace_abstract, Module, UniqueRepresentation

Module of (Hecke) weakly holomorphic modular forms for the given group, base ring, weight and multiplier

class sage.modular.modform_hecketriangle.space.ZeroForm(group, base_ring, k, ep, n)
Bases: FormsSpace_abstract, Module, UniqueRepresentation

Zero Module for the zero form for the given group, base ring weight and multiplier

coordinate_vector(v)

Return the coordinate vector of v with respect to the basis self.gens().

Since this is the zero module which only contains the zero form the trivial vector in the trivial module of dimension 0 is returned.

INPUT:

• v – An element of self, i.e. in this case the zero vector.

EXAMPLES:

sage: from sage.modular.modform_hecketriangle.space import ZeroForm
sage: MF = ZeroForm(6, QQ, 3, -1)
sage: el = MF(0)
sage: el
O(q^5)
sage: vec = el.coordinate_vector()
sage: vec
()
sage: vec.parent()
Vector space of dimension 0 over Fraction Field of Univariate Polynomial Ring in d over Rational Field
sage: vec.parent() == MF.module()
True

dimension()

Return the dimension of self. Since this is the zero module 0 is returned.

EXAMPLES:
sage: from sage.modular.modform_hecketriangle.space import ZeroForm
sage: ZeroForm(6, CC, 3, -1).dimension()
0

gens()
Return a basis of self as a list of basis elements. Since this is the zero module an empty list is returned.

EXAMPLES:

sage: from sage.modular.modform_hecketriangle.space import ZeroForm
sage: ZeroForm(6, CC, 3, -1).gens()
[]

sage.modular.modform_hecketriangle.space.canonical_parameters(group, base_ring, k, ep, n=None)
Return a canonical version of the parameters.

EXAMPLES:

sage: from sage.modular.modform_hecketriangle.space import canonical_parameters
sage: canonical_parameters(5, ZZ, 20/3, int(1))
(Hecke triangle group for n = 5, Integer Ring, 20/3, 1, 5)
sage: canonical_parameters(infinity, ZZ, 2, int(-1))
(Hecke triangle group for n = +Infinity, Integer Ring, 2, -1, +Infinity)

2.13 Subspaces of modular forms for Hecke triangle groups

AUTHORS:

• Jonas Jermann (2013): initial version

sage.modular.modform_hecketriangle.subspace.ModularFormsSubSpace(*args, **kwargs)
Create a modular forms subspace generated by the supplied arguments if possible. Instead of a list of generators also multiple input arguments can be used. If reduce=True then the corresponding ambient space is choosen as small as possible. If no subspace is available then the ambient space is returned.

EXAMPLES:

sage: from sage.modular.modform_hecketriangle.subspace import ModularFormsSubSpace
sage: from sage.modular.modform_hecketriangle.space import ModularForms
sage: MF = ModularForms()
sage: subspace = ModularFormsSubSpace(MF.E4()^3, MF.E6()^2+MF.Delta(), MF.Delta())
sage: subspace
Subspace of dimension 2 of ModularForms(n=3, k=12, ep=1) over Integer Ring
sage: subspace.ambient_space()
ModularForms(n=3, k=12, ep=1) over Integer Ring
sage: subspace.gens()
[1 + 720*q + 179280*q^2 + 16954560*q^3 + 396974160*q^4 + O(q^5), 1 - 1007*q + 220728*q^2 + 16519356*q^3 + 399516304*q^4 + O(q^5)]
sage: ModularFormsSubSpace(MF.E4()^3-MF.E6()^2, reduce=True).ambient_space()
CuspForms(n=3, k=12, ep=1) over Integer Ring
sage: ModularFormsSubSpace(MF.E4()^3-MF.E6()^2, MF.J_inv()*MF.E4()^3, reduce=True)
WeakModularForms(n=3, k=12, ep=1) over Integer Ring

266 Chapter 2. Modular Forms for Hecke Triangle Groups
Submodule of (Hecke) forms in the given ambient space for the given basis.

**basis()**

Return the basis of *self* in the ambient space.

**EXAMPLES:**

```python
given code```

**change_ambient_space(new_ambient_space)**

Return a new subspace with the same basis but inside a different ambient space (if possible).

**EXAMPLES:**

```python
given code```

**change_ring(new_base_ring)**

Return the same space as *self* but over a new base ring *new_base_ring*.

**EXAMPLES:**

```python
given code```

**contains_coeff_ring()**

Return whether *self* contains its coefficient ring.

**EXAMPLES:**

```python
given code```
coordinate_vector(v)

Return the coordinate vector of v with respect to the basis self.gens().

INPUT:

• v – An element of self.

OUTPUT:

The coordinate vector of v with respect to the basis self.gens().

Note: The coordinate vector is not an element of self.module().

EXAMPLES:

```sage
from sage.modular.modform_hecketriangle.space import ModularForms, QuasiCuspForms
MF = ModularForms(n=6, k=20, ep=1)
space = MF.subspace([MF.Delta()*MF.E4()^2, MF.gen(0)])
space.coordinate_vector(MF.gen(0) + MF.Delta()*MF.E4()^2).parent()
Vector space of dimension 2 over Fraction Field of Univariate Polynomial Ring
in d over Integer Ring
```

```sage
MF = ModularForms(n=4, k=24, ep=-1)
space = MF.subspace([MF.gen(0), MF.gen(2)])
space.coordinate_vector(MF.Delta()^2).parent()
Vector space of dimension 2 over Fraction Field of Univariate Polynomial Ring
in d over Integer Ring
```

```sage
MF = QuasiCuspForms(n=infinity, k=12, ep=1)
space = MF.subspace([MF.Delta(), MF.E4()*MF.f_inf()*MF.E2()*MF.f_i(), MF.E4()*MF.f_inf()*MF.E2()^2, MF.E4()*MF.f_inf()*(MF.E4()-MF.E2()^2)])
el = MF.E4()*MF.f_inf()*(7*MF.E4() - 3*MF.E2()^2)
space.coordinate_vector(el)
```

degree()

Return the degree of self.

EXAMPLES:
sage: from sage.modular.modform_hecketriangle.space import ModularForms
sage: MF = ModularForms(n=6, k=20, ep=1)
sage: subspace = MF.subspace([(MF.Delta()*MF.E4()^2).as_ring_element(), MF.gen(0)])
sage: subspace.degree()
4
sage: subspace.degree() == subspace.ambient_space().degree()
True

dimension()
Return the dimension of self.
EXAMPLES:

sage: from sage.modular.modform_hecketriangle.space import ModularForms
sage: MF = ModularForms(n=6, k=20, ep=1)
sage: subspace = MF.subspace([(MF.Delta()*MF.E4()^2).as_ring_element(), MF.gen(0)])
sage: subspace.dimension()
2
sage: subspace.dimension() == len(subspace.gens())
True

gens()
Return the basis of self.
EXAMPLES:

sage: from sage.modular.modform_hecketriangle.space import ModularForms
sage: MF = ModularForms(n=6, k=20, ep=1)
sage: subspace = MF.subspace([(MF.Delta()*MF.E4()^2).as_ring_element(), MF.gen(0)])
sage: subspace.gens()
[q + 78*q^2 + 2781*q^3 + 59812*q^4 + O(q^5), 1 + 360360*q^4 + O(q^5)]
sage: subspace.gens()[0].parent() == subspace
True

rank()
Return the rank of self.
EXAMPLES:

sage: from sage.modular.modform_hecketriangle.space import ModularForms
sage: MF = ModularForms(n=6, k=20, ep=1)
sage: subspace = MF.subspace([(MF.Delta()*MF.E4()^2).as_ring_element(), MF.gen(0)])
sage: subspace.rank()
2
sage: subspace.rank() == subspace.dimension()
True

sage.modular.modform_hecketriangle.subspace.canonical_parameters(ambient_space, basis, check=True)

Return a canonical version of the parameters. In particular the list/tuple basis is replaced by a tuple of linearly independent elements in the ambient space.
If check=False (default: True) then basis is assumed to already be a basis.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.subspace import canonical_parameters
tsage: from sage.modular.modform_hecketriangle.space import ModularForms
sage: MF = ModularForms(n=6, k=12, ep=1)
sage: canonical_parameters(MF, [MF.Delta().as_ring_element(), MF.gen(0), 2*MF.gen(0)])
(ModularForms(n=6, k=12, ep=1) over Integer Ring, (q + 30*q^2 + 333*q^3 + 1444*q^4 + O(q^5), 1 + 26208*q^3 + 530712*q^4 + O(q^5)))
```

### 2.14 Series constructor for modular forms for Hecke triangle groups

AUTHORS:

- Based on the thesis of John Garrett Leo (2008)
- Jonas Jermann (2013): initial version

**Note:** \( J\_inv\_ZZ \) is the main function used to determine all Fourier expansions.

class sage.modular.modform_hecketriangle.series_constructor.MFSeriesConstructor(group, 
prec)

Bases: SageObject, UniqueRepresentation

Constructor for the Fourier expansion of some (specific, basic) modular forms.

The constructor is used by forms elements in case their Fourier expansion is needed or requested.

**Delta_ZZ()**

Return the rational Fourier expansion of Delta, where the parameter \( d \) is replaced by 1.

**Note:** The Fourier expansion of Delta for \( d! = 1 \) is given by \( d * \text{Delta}_ZZ(q/d) \).

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.series_constructor import MFSeriesConstructor
sage: MFSeriesConstructor(prec=3).Delta_ZZ()
q - 1/72*q^2 + 7/82944*q^3 + O(q^4)
sage: MFSeriesConstructor(group=5, prec=3).Delta_ZZ()
q + 47/200*q^2 + 11367/640000*q^3 + O(q^4)
sage: MFSeriesConstructor(group=infinity, prec=3).Delta_ZZ()
Power Series Ring in q over Rational Field
sage: MFSeriesConstructor(group=infinity, prec=3).Delta_ZZ()
q + 3/8*q^2 + 63/1024*q^3 + O(q^4)
```

**E2_ZZ()**

Return the rational Fourier expansion of E2, where the parameter \( d \) is replaced by 1.
Note: The Fourier expansion of $E_2$ for $d=1$ is given by $E_2_{\mathbb{Z}}(q/d)$.

EXAMPLES:

```python
sage: from sage.modular.modform_hecketriangle.series_constructor import MFSeriesConstructor
sage: MFSeriesConstructor(prec=3).E2_ZZ()
1 - 1/72*q - 1/41472*q^2 + O(q^3)
sage: MFSeriesConstructor(group=5, prec=3).E2_ZZ()
1 - 9/200*q - 369/320000*q^2 + O(q^3)
sage: MFSeriesConstructor(group=5, prec=3).E2_ZZ().parent()
Power Series Ring in q over Rational Field
sage: MFSeriesConstructor(group=infinity, prec=3).E2_ZZ()
1 - 1/8*q - 1/512*q^2 + O(q^3)
```

$E_4_{\mathbb{Z}}()$

Return the rational Fourier expansion of $E_4$, where the parameter $d$ is replaced by 1.

Note: The Fourier expansion of $E_4$ for $d=1$ is given by $E_4_{\mathbb{Z}}(q/d)$.

EXAMPLES:

```python
sage: from sage.modular.modform_hecketriangle.series_constructor import MFSeriesConstructor
sage: MFSeriesConstructor(prec=3).E4_ZZ()
1 + 5/36*q + 5/6912*q^2 + O(q^3)
sage: MFSeriesConstructor(group=5, prec=3).E4_ZZ()
1 + 21/100*q + 483/32000*q^2 + O(q^3)
sage: MFSeriesConstructor(group=5, prec=3).E4_ZZ().parent()
Power Series Ring in q over Rational Field
sage: MFSeriesConstructor(group=infinity, prec=3).E4_ZZ()
1 + 1/4*q + 7/256*q^2 + O(q^3)
```

$E_6_{\mathbb{Z}}()$

Return the rational Fourier expansion of $E_6$, where the parameter $d$ is replaced by 1.

Note: The Fourier expansion of $E_6$ for $d=1$ is given by $E_6_{\mathbb{Z}}(q/d)$.

EXAMPLES:

```python
sage: from sage.modular.modform_hecketriangle.series_constructor import MFSeriesConstructor
sage: MFSeriesConstructor(prec=3).E6_ZZ()
1 - 7/24*q - 77/13824*q^2 + O(q^3)
sage: MFSeriesConstructor(group=5, prec=3).E6_ZZ()
1 - 37/200*q - 14663/320000*q^2 + O(q^3)
sage: MFSeriesConstructor(group=5, prec=3).E6_ZZ().parent()
Power Series Ring in q over Rational Field
```

(continues on next page)
EisensteinSeries_ZZ(k)

Return the rational Fourier expansion of the normalized Eisenstein series of weight \( k \), where the parameter \( d \) is replaced by 1.

Only arithmetic groups with \( n < \infty \) are supported!

**Note:** The Fourier expansion of the series is given by \( \text{EisensteinSeries}_\mathbb{Z}(q/d) \).

**INPUT:**

- \( k \) – A non-negative even integer, namely the weight.

**EXAMPLES:**

\[
\begin{align*}
\text{sage: } & \text{from sage.modular.modform hecketriangle.series_constructor import } \backarrow\text{MFSeriesConstructor} \\
\text{sage: } & \text{MFC = MFSeriesConstructor(prec=6)} \\
\text{sage: } & \text{MFC.EisensteinSeries_ZZ(k=0)} \\
& 1 \\
\text{sage: } & \text{MFC.EisensteinSeries_ZZ(k=2)} \\
& 1 - 1/72*q - 1/41472*q^2 - 1/53747712*q^3 - 7/371504185344*q^4 - 1/106993205379072*q^5 + O(q^6) \\
\text{sage: } & \text{MFC.EisensteinSeries_ZZ(k=6)} \\
& 1 - 7/24*q - 77/13824*q^2 - 427/17915904*q^3 - 7399/123834728448*q^4 - 3647/35664401793024*q^5 + O(q^6) \\
\text{sage: } & \text{MFC.EisensteinSeries_ZZ(k=12)} \\
& 1 + 455/8292*q + 310765/4776192*q^2 + 20150585/6189944832*q^3 + 1909340615/12322650819489792*q^4 + O(q^6) \\
\text{sage: } & \text{MFC.EisensteinSeries_ZZ(k=12).parent()} \\
& \text{Power Series Ring in q over Rational Field} \\
\text{sage: } & \text{MFC = MFSeriesConstructor(group=4, prec=5)} \\
\text{sage: } & \text{MFC.EisensteinSeries_ZZ(k=2)} \\
& 1 - 1/32*q - 5/8192*q^2 - 1/524288*q^3 - 13/536870912*q^4 + O(q^5) \\
\text{sage: } & \text{MFC.EisensteinSeries_ZZ(k=4)} \\
& 1 + 3/16*q + 39/4096*q^2 + 21/262144*q^3 + 327/268435456*q^4 + O(q^5) \\
\text{sage: } & \text{MFC.EisensteinSeries_ZZ(k=6)} \\
& 1 - 7/32*q - 287/8192*q^2 - 427/524288*q^3 - 9247/536870912*q^4 + O(q^5) \\
\text{sage: } & \text{MFC.EisensteinSeries_ZZ(k=12)} \\
& 1 + 63/11056*q + 133119/2830336*q^2 + 2790001/181141504*q^3 + 272631807/185488900096*q^4 + O(q^5) \\
\text{sage: } & \text{MFC = MFSeriesConstructor(group=6, prec=5)} \\
\text{sage: } & \text{MFC.EisensteinSeries_ZZ(k=2)} \\
& 1 - 1/18*q - 1/648*q^2 - 7/209952*q^3 - 7/22674816*q^4 + O(q^5) \\
\text{sage: } & \text{MFC.EisensteinSeries_ZZ(k=4)} \\
& 1 + 2/9*q + 1/54*q^2 + 37/52488*q^3 + 73/5668704*q^4 + O(q^5) \\
\text{sage: } & \text{MFC.EisensteinSeries_ZZ(k=6)} \\
\end{align*}
\]
1 \text{-} 1/6*q - 11/216*q^2 - 271/69984*q^3 - 1057/7558272*q^4 + O(q^5)

\textcolor{green}{\textbf{sage: MFC.EisensteinSeries_ZZ(k=12)}}

1 + 182/151329*q + 62153/2723922*q^2 + 16186807/882550728*q^3 + 381868123/
\rightarrow 95315478624*q^4 + O(q^5)

\textit{G\_inv\_ZZ()} 

Return the rational Fourier expansion of \textit{G\_inv}, where the parameter \textit{d} is replaced by 1.

\textbf{Note: } The Fourier expansion of \textit{G\_inv} for \textit{d}!=1 is given by \textit{d*G\_inv\_ZZ}(q/d).

\textbf{EXAMPLES:}

\textcolor{green}{\textbf{sage: from sage.modular.modform_hecketriangle.series_constructor import \_\_MFSeriesConstructor}}

\textcolor{green}{\textbf{sage: MFSeriesConstructor(group=4, prec=3).G\_inv\_ZZ()}}

q\textsuperscript{-}1 - 3/32 - 955/16384*q + O(q^2)

\textcolor{green}{\textbf{sage: MFSeriesConstructor(group=8, prec=3).G\_inv\_ZZ()}}

q\textsuperscript{-}1 - 15/128 - 15139/262144*q + O(q^2)

\textcolor{green}{\textbf{sage: MFSeriesConstructor(group=infinity, prec=3).G\_inv\_ZZ()}}

Laurent Series Ring in \textit{q} over Rational Field

\textcolor{green}{\textbf{sage: MFSeriesConstructor(group=infinity, prec=3).G\_inv\_ZZ()}}

q\textsuperscript{-}1 - 1/8 - 59/1024*q + O(q^2)

\textit{J\_inv\_ZZ()} 

Return the rational Fourier expansion of \textit{J\_inv}, where the parameter \textit{d} is replaced by 1.

This is the main function used to determine all Fourier expansions!

\textbf{Note: } The Fourier expansion of \textit{J\_inv} for \textit{d}!=1 is given by \textit{J\_inv\_ZZ}(q/d).

\textbf{Todo: } The functions that are used in this implementation are products of hypergeometric series with other, elementary, functions. Implement them and clean up this representation.

\textbf{EXAMPLES:}

\textcolor{green}{\textbf{sage: from sage.modular.modform_hecketriangle.series_constructor import \_\_MFSeriesConstructor}}

\textcolor{green}{\textbf{sage: MFSeriesConstructor(group=5, prec=3).J\_inv\_ZZ()}}

q\textsuperscript{-}1 + 79/200 + 42877/640000*q + O(q^2)

\textcolor{green}{\textbf{sage: MFSeriesConstructor(group=5, prec=3).J\_inv\_ZZ().parent()}}

Laurent Series Ring in \textit{q} over Rational Field

\textcolor{green}{\textbf{sage: MFSeriesConstructor(group=infinity, prec=3).J\_inv\_ZZ()}}

q\textsuperscript{-}1 + 3/8 + 69/1024*q + O(q^2)

\textit{f\_i\_ZZ()} 

Return the rational Fourier expansion of \textit{f\_i}, where the parameter \textit{d} is replaced by 1.
Note: The Fourier expansion of $f_i$ for $d=1$ is given by $f_i_{\mathbb{Z}}(q/d)$.

EXAMPLES:

sage: from sage.modular.modform_hecketriangle.series_constructor import MFSeriesConstructor
sage: MFSeriesConstructor(prec=3).f_i_{\mathbb{Z}}()
1 - 7/24*q - 77/13824*q^2 + O(q^3)
sage: MFSeriesConstructor(group=5, prec=3).f_i_{\mathbb{Z}}()
1 - 13/40*q - 351/64000*q^2 + O(q^3)
sage: MFSeriesConstructor(group=5, prec=3).f_i_{\mathbb{Z}}().parent()
Power Series Ring in q over Rational Field
sage: MFSeriesConstructor(group=infinity, prec=3).f_i_{\mathbb{Z}}()
1 - 3/8*q + 3/512*q^2 + O(q^3)

$f_{\infty_{\mathbb{Z}}}$

Return the rational Fourier expansion of $f_{\infty}$, where the parameter $d$ is replaced by $1$.

Note: The Fourier expansion of $f_{\infty}$ for $d=1$ is given by $d*f_{\infty_{\mathbb{Z}}}(q/d)$.

EXAMPLES:

sage: from sage.modular.modform_hecketriangle.series_constructor import MFSeriesConstructor
sage: MFSeriesConstructor(prec=3).f_{\infty_{\mathbb{Z}}}
q - 1/72*q^2 + 7/82944*q^3 + O(q^4)
sage: MFSeriesConstructor(group=5, prec=3).f_{\infty_{\mathbb{Z}}}
q - 9/200*q^2 + 279/640000*q^3 + O(q^4)
sage: MFSeriesConstructor(group=5, prec=3).f_{\infty_{\mathbb{Z}}}.parent()
Power Series Ring in q over Rational Field
sage: MFSeriesConstructor(group=infinity, prec=3).f_{\infty_{\mathbb{Z}}}
q - 1/8*q^2 + 7/1024*q^3 + O(q^4)

$f_{\rho_{\mathbb{Z}}}$

Return the rational Fourier expansion of $f_{\rho}$, where the parameter $d$ is replaced by $1$.

Note: The Fourier expansion of $f_{\rho}$ for $d=1$ is given by $f_{\rho_{\mathbb{Z}}}(q/d)$.

EXAMPLES:

sage: from sage.modular.modform_hecketriangle.series_constructor import MFSeriesConstructor
sage: MFSeriesConstructor(prec=3).f_{\rho_{\mathbb{Z}}}
1 + 5/36*q + 5/6912*q^2 + O(q^3)
sage: MFSeriesConstructor(group=5, prec=3).f_{\rho_{\mathbb{Z}}}
1 + 7/100*q + 21/160000*q^2 + O(q^3)
sage: MFSeriesConstructor(group=5, prec=3).f_{\rho_{\mathbb{Z}}}.parent()
Power Series Ring in q over Rational Field
sage: MFSeriesConstructor(group=infinity, prec=3).f_rho_ZZ()
1

group()
Return the (Hecke triangle) group of self.

EXAMPLES:

sage: from sage.modular.modform_hecketriangle.series_constructor import MFSeriesConstructor
sage: MFSeriesConstructor(group=4).group()
Hecke triangle group for n = 4

hecke_n()
Return the parameter n of the (Hecke triangle) group of self.

EXAMPLES:

sage: from sage.modular.modform_hecketriangle.series_constructor import MFSeriesConstructor
sage: MFSeriesConstructor(group=4).hecke_n()
4

prec()
Return the used default precision for the PowerSeriesRing or LaurentSeriesRing.

EXAMPLES:

sage: from sage.modular.modform_hecketriangle.series_constructor import MFSeriesConstructor
sage: MFSeriesConstructor(group=5).prec()
10
sage: MFSeriesConstructor(group=5, prec=20).prec()
20
3.1 Graded quasimodular forms ring

Let $E_2$ be the weight 2 Eisenstein series defined by

$$E_2(z) = 1 - \frac{2k}{B_k} \sum_{n=1}^{\infty} \sigma(n)q^n$$

where $\sigma$ is the sum of divisors function and $q = \exp(2\pi iz)$ is the classical parameter at infinity, with $\text{im}(z) > 0$. This weight 2 Eisenstein series is not a modular form as it does not satisfy the modularity condition:

$$z^2E_2(-1/z) = E_2(z) + \frac{2k}{4\pi i B_k z}.$$  

$E_2$ is a quasimodular form of weight 2. General quasimodular forms of given weight can also be defined. We denote by $QM$ the graded ring of quasimodular forms for the full modular group $SL_2(\mathbb{Z})$.

The SageMath implementation of the graded ring of quasimodular forms uses the following isomorphism:

$$QM \cong M_*[E_2]$$

where $M_* \cong \mathbb{C}[E_4, E_6]$ is the graded ring of modular forms for $SL_2(\mathbb{Z})$. (see `sage.modular.modform.ring.ModularFormsRing`).

More generally, if $\Gamma \leq SL_2(\mathbb{Z})$ is a congruence subgroup, then the graded ring of quasimodular forms for $\Gamma$ is given by $M_*^\Gamma[E_2]$ where $M_*^\Gamma$ is the ring of modular forms for $\Gamma$.

The SageMath implementation of the graded quasimodular forms ring allows computation of a set of generators and perform usual arithmetic operations.

EXAMPLES:

```
sage: QM = QuasiModularForms(1); QM
Ring of Quasimodular Forms for Modular Group SL(2,Z) over Rational Field
dsage: QM.gens()
[1 - 24*q - 72*q^2 - 96*q^3 - 168*q^4 - 144*q^5 + O(q^6),
 1 + 240*q + 2160*q^2 + 6720*q^3 + 17520*q^4 + 30240*q^5 + O(q^6),
 1 - 504*q - 16632*q^2 - 122976*q^3 - 532728*q^4 - 1575504*q^5 + O(q^6)]
sage: E2 = QM.0; E4 = QM.1; E6 = QM.2
sage: E2 * E4 + E6
2 - 288*q - 20304*q^2 - 185472*q^3 - 855216*q^4 - 2697408*q^5 + O(q^6)
sage: E2.parent()
Ring of Quasimodular Forms for Modular Group SL(2,Z) over Rational Field
```
The `polygen` method also return the weight-2 Eisenstein series as a polynomial variable over the ring of modular forms:

```
sage: QM = QuasiModularForms(1)
sage: E2 = QM.polygen(); E2
E2
sage: E2.parent()
Univariate Polynomial Ring in E2 over Ring of Modular Forms for Modular Group SL(2,Z)
˓→over Rational Field
```

An element of a ring of quasimodular forms can be created via a list of modular forms or graded modular forms. The $i$-th index of the list will correspond to the $i$-th coefficient of the polynomial in $E_2$:

```
sage: QM = QuasiModularForms(1)
sage: E2 = QM.0
sage: Delta = CuspForms(1, 12).0
sage: E4 = ModularForms(1, 4).0
sage: F = QM([Delta, E4, Delta + E4]); F
2 + 410*q - 12696*q^2 + 50424*q^3 + 1076264*q^4 + 10431996*q^5 + O(q^6)
sage: F == Delta + E4 * E2 + (Delta + E4) * E2^2
True
```

One may also create rings of quasimodular forms for certain congruence subgroups:

```
sage: QM = QuasiModularForms(Gamma0(5)); QM
Ring of Quasimodular Forms for Congruence Subgroup Gamma0(5) over Rational Field
sage: QM.ngens()
4
```

The first generator is the weight 2 Eisenstein series:

```
sage: E2 = QM.0; E2
1 - 24*q - 72*q^2 - 96*q^3 - 168*q^4 - 144*q^5 + O(q^6)
```

The other generators correspond to the generators given by the method `sage.modular.modform.ring.ModularFormsRing.gens()`:

```
sage: QM.gens()
[1 - 24*q - 72*q^2 - 96*q^3 - 168*q^4 - 144*q^5 + O(q^6),
 1 + 6*q + 18*q^2 + 24*q^3 + 42*q^4 + 6*q^5 + O(q^6),
 1 + 240*q^5 + O(q^6),
 q + 10*q^3 + 28*q^4 + 35*q^5 + O(q^6)]
sage: QM.modular_forms_subring().gens()
[1 + 6*q + 18*q^2 + 24*q^3 + 42*q^4 + 6*q^5 + O(q^6),
 1 + 240*q^5 + O(q^6),
 q + 10*q^3 + 28*q^4 + 35*q^5 + O(q^6)]
```

It is possible to convert a graded quasimodular form into a polynomial where each variable corresponds to a generator of the ring:

```
sage: QM = QuasiModularForms(1)
sage: E2, E4, E6 = QM.gens()
sage: F = E2*E4*E6 + E6^2; F
2 - 1296*q + 91584*q^2 + 14591808*q^3 + 464670432*q^4 + 6160281120*q^5 + O(q^6)
```

(continues on next page)
The generators of the polynomial ring have degree equal to the weight of the corresponding form:

```
sage: P.inject_variables()
Defining E2, E4, E6
sage: E2.degree()
2
sage: E4.degree()
4
sage: E6.degree()
6
```

This works also for congruence subgroup:

```
sage: QM = QuasiModularForms(Gamma1(4))
sage: QM.ngens()
5
sage: QM.polynomial_ring()
Multivariate Polynomial Ring in E2, E2_0, E2_1, E3_0, E3_1 over Rational Field
sage: (QM.0 + QM.1*QM.0^2 + QM.3 + QM.4^3).polynomial()
E3_1^3 + E2^2*E2_0 + E3_0 + E2
```

One can also convert a multivariate polynomial into a quasimodular form:

```
sage: QM.polynomial_ring().inject_variables()
Defining E2, E2_0, E2_1, E3_0, E3_1
sage: QM.from_polynomial(E3_1^3 + E2^2*E2_0 + E3_0 + E2)
3 - 72*q + 396*q^2 + 2081*q^3 + 19752*q^4 + 98712*q^5 + O(q^6)
```

Note:
- Currently, the only supported base ring is the Rational Field;
- Spaces of quasimodular forms of fixed weight are not yet implemented.

REFERENCE:
See section 5.3 (page 58) of [Zag2008]

AUTHORS:
- David Ayotte (2021-03-18): initial version

class sage.modular.quasimodform.ring.QuasiModularForms(group=1, base_ring=Rational Field, name='E2')

Bases: Parent, UniqueRepresentation

The graded ring of quasimodular forms for the full modular group \( \text{SL}_2(\mathbb{Z}) \), with coefficients in a ring.

EXAMPES:
sage: QM = QuasiModularForms(1); QM
Ring of Quasimodular Forms for Modular Group SL(2,Z) over Rational Field
sage: QM.gens()
[1 - 24*q - 72*q^2 - 96*q^3 - 168*q^4 - 144*q^5 + O(q^6),
 1 + 240*q + 2160*q^2 + 6720*q^3 + 17520*q^4 + 30240*q^5 + O(q^6),
 1 - 504*q - 16632*q^2 - 122976*q^3 - 532728*q^4 - 1575504*q^5 + O(q^6)]

It is possible to access the weight 2 Eisenstein series:

sage: QM.weight_2_eisenstein_series()
1 - 24*q - 72*q^2 - 96*q^3 - 144*q^4 - 144*q^5 + O(q^6)

Currently, the only supported base ring is the rational numbers:

sage: QuasiModularForms(1, GF(5))
Traceback (most recent call last):
... NotImplementedError: base ring other than Q are not yet supported for quasimodular forms ring

Element

... alias of QuasiModularFormsElement

from_polynomial(polynomial)

Convert the given polynomial \( P(x, ..., y) \) to the graded quasiform \( P(g_0, ..., g_n) \) where the \( g_i \) are the generators given by gens().

INPUT:

- polynomial – A multivariate polynomial

OUTPUT: the graded quasimodular forms \( P(g_0, ..., g_n) \)

EXAMPLES:

sage: QM = QuasiModularForms(1)
sage: P.<x, y, z> = QQ[]
sage: QM.from_polynomial(x)
1 - 24*q - 72*q^2 - 96*q^3 - 168*q^4 - 144*q^5 + O(q^6)
sage: QM.from_polynomial(x) == QM.0
True
sage: QM.from_polynomial(y) == QM.1
True
sage: QM.from_polynomial(z) == QM.2
True
sage: QM.from_polynomial(x^2 + y + x*z + 1)
4 - 336*q - 2016*q^2 + 322368*q^3 + 3691392*q^4 + 21797280*q^5 + O(q^6)
sage: QM = QuasiModularForms(Gamma0(2))
sage: P = QM.polynomial_ring()
sage: P.inject_variables()
Defining E2, E2_0, E4_0
sage: QM.from_polynomial(E2)
1 - 24*q - 72*q^2 - 96*q^3 - 168*q^4 - 144*q^5 + O(q^6)
sage: QM.from_polynomial(E2 + E4_0*E2_0) == QM.0 + QM.2*QM.1
True
Naturally, the number of variable must not exceed the number of generators:

```
sage: P = PolynomialRing(QQ, 'F', 4)
sage: P.inject_variables()
Defining F0, F1, F2, F3
sage: QM.from_polynomial(F0 + F1 + F2 + F3)
Traceback (most recent call last):
...
ValueError: the number of variables (4) of the given polynomial cannot exceed the number of generators (3) of the quasimodular forms ring
```

\textbf{gen}(n)

Return the \textit{n}-th generator of the quasimodular forms ring.

\textbf{EXAMPLES:}

```
sage: QM = QuasiModularForms(1)
sage: QM.0
1 - 24*q - 72*q^2 - 96*q^3 - 168*q^4 - 144*q^5 + O(q^6)
sage: QM.1
1 + 240*q + 2160*q^2 + 6720*q^3 + 17520*q^4 + 30240*q^5 + O(q^6)
sage: QM.2
1 - 504*q - 16632*q^2 - 122976*q^3 - 532728*q^4 - 1575504*q^5 + O(q^6)
sage: QM = QuasiModularForms(5)
sage: QM.0
1 - 24*q - 72*q^2 - 96*q^3 - 168*q^4 - 144*q^5 + O(q^6)
sage: QM.1
1 + 6*q + 18*q^2 + 24*q^3 + 42*q^4 + 6*q^5 + O(q^6)
sage: QM.2
1 + 240*q^5 + O(q^6)
sage: QM.3
q + 10*q^3 + 28*q^4 + 35*q^5 + O(q^6)
sage: QM.4
Traceback (most recent call last):
...
IndexError: list index out of range
```

\textbf{generators()}

Return a list of generators of the quasimodular forms ring.

Note that the generators of the modular forms subring are the one given by the method \texttt{sage.modular.modform.ring.ModularFormsRing.gen_forms()}

\textbf{EXAMPLES:}

```
sage: QM = QuasiModularForms(1)
sage: QM.gens()

[1 - 24*q - 72*q^2 - 96*q^3 - 168*q^4 - 144*q^5 + O(q^6),
1 + 240*q + 2160*q^2 + 6720*q^3 + 17520*q^4 + 30240*q^5 + O(q^6),
1 - 504*q - 16632*q^2 - 122976*q^3 - 532728*q^4 - 1575504*q^5 + O(q^6)]
sage: QM.modular_forms_subring().gen_forms()

[1 + 240*q + 2160*q^2 + 6720*q^3 + 17520*q^4 + 30240*q^5 + O(q^6),
1 - 504*q - 16632*q^2 - 122976*q^3 - 532728*q^4 - 1575504*q^5 + O(q^6)]
sage: QM = QuasiModularForms(5)
sage: QM.gens()
```

(continues on next page)
An alias of this method is `generators`:

```
sage: QuasiModularForms(1).generators()
[1 - 24*q - 72*q^2 - 96*q^3 - 168*q^4 - 144*q^5 + O(q^6),
 1 + 6*q + 18*q^2 + 24*q^3 + 42*q^4 + 6*q^5 + O(q^6),
 1 + 240*q^5 + O(q^6),
 q + 10*q^3 + 28*q^4 + 35*q^5 + O(q^6)]
```

gens()

Return a list of generators of the quasimodular forms ring.

Note that the generators of the modular forms subring are the one given by the method `sage.modular.modform.ring.ModularFormsRing.gen_forms()`

EXAMPLES:

```
sage: QM = QuasiModularForms(1)
sage: QM.gens()
[1 - 24*q - 72*q^2 - 96*q^3 - 168*q^4 - 144*q^5 + O(q^6),
 1 + 6*q + 18*q^2 + 24*q^3 + 42*q^4 + 6*q^5 + O(q^6),
 1 + 240*q^5 + O(q^6),
 q + 10*q^3 + 28*q^4 + 35*q^5 + O(q^6)]
```

An alias of this method is `generators`:

```
sage: QuasiModularForms(1).generators()
[1 - 24*q - 72*q^2 - 96*q^3 - 168*q^4 - 144*q^5 + O(q^6),
 1 + 6*q + 18*q^2 + 24*q^3 + 42*q^4 + 6*q^5 + O(q^6),
 1 + 240*q^5 + O(q^6),
 q + 10*q^3 + 28*q^4 + 35*q^5 + O(q^6)]
```

group()

Return the congruence subgroup attached to the given quasimodular forms ring.

EXAMPLES:

```
sage: QM = QuasiModularForms(1)
sage: QM.group()
Modular Group SL(2,Z)
sage: QM.group() is SL2Z
True
```

(continues on next page)
modular_forms_of_weight(weight)

Return the space of modular forms on this group of the given weight.

EXAMPLES:

```
sage: QM = QuasiModularForms(1)
sage: QM.modular_forms_of_weight(12)
Modular Forms space of dimension 2 for Modular Group SL(2,Z) of weight 12 over Rational Field

sage: QM = QuasiModularForms(Gamma1(3))
sage: QM.modular_forms_of_weight(4)
Modular Forms space of dimension 2 for Congruence Subgroup Gamma1(3) of weight 4 over Rational Field
```

modular_forms_subring()

Return the subring of modular forms of this ring of quasimodular forms.

EXAMPLES:

```
sage: QuasiModularForms(1).modular_forms_subring()
Ring of Modular Forms for Modular Group SL(2,Z) over Rational Field

sage: QuasiModularForms(5).modular_forms_subring()
Ring of Modular Forms for Congruence Subgroup Gamma0(5) over Rational Field
```

ngens()

Return the number of generators of the given graded quasimodular forms ring.

EXAMPLES:

```
sage: QuasiModularForms(1).ngens()
3
```

one()

Return the one element of this ring.

EXAMPLES:

```
sage: QM = QuasiModularForms(1)
sage: QM.one()
1
sage: QM.one().is_one()
True
```

polygen()

Return the generator of this quasimodular form space as a polynomial ring over the modular form subring.

Note that this generator correspond to the weight-2 Eisenstein series. The default name of this generator is E2.

EXAMPLES:
polynomial_ring(names=None)

Return a multivariate polynomial ring of which the quasimodular forms ring is a quotient.

In the case of the full modular group, this ring is $R[E_2, E_4, E_6]$ where $E_2$, $E_4$ and $E_6$ have degrees 2, 4 and 6 respectively.

INPUT:

- names (str, default: None) – a list or tuple of names (strings), or a comma separated string. Defines the names for the generators of the multivariate polynomial ring. The default names are of the following form:
  - $E_2$ denotes the weight 2 Eisenstein series;
  - $E_{k,i}$ and $S_{k,i}$ denote the $i$-th basis element of the weight $k$ Eisenstein subspace and cuspidal subspace respectively;
  - If the level is one, the default names are $E_2$, $E_4$ and $E_6$;
  - In any other cases, we use the letters $F_k$, $G_k$, $H_k$, $...$, $F_{F_k}$, $F_{G_k}$, $...$ to denote any generator of weight $k$.

OUTPUT: A multivariate polynomial ring in the variables names

EXAMPLES:

```
sage: QM = QuasiModularForms(1)
sage: QM = QuasiModularForms(1)
sage: QM.polynomial_ring(); P
Multivariate Polynomial Ring in E2, E4, E6 over Rational Field
sage: P.inject_variables()
Defining E2, E4, E6
sage: E2.degree()
2
sage: E4.degree()
4
sage: E6.degree()
6
```

Example when the level is not one:

```
sage: QM = QuasiModularForms(Gamma0(29))
sage: P_29 = QM.polynomial_ring()
sage: P_29
Multivariate Polynomial Ring in E2, F2, S2_0, S2_1, E4_0, F4, G4, H4 over Rational Field
sage: P_29.inject_variables()
Defining E2, F2, S2_0, S2_1, E4_0, F4, G4, H4
sage: F2.degree()
(continues on next page)```
The name $S_k_i$ stands for the $i$-th basis element of the cuspidal subspace of weight $k$:

\begin{verbatim}
 sage: F2 = QM.from_polynomial(S2_0)
sage: F2.qexp(10)
q - q^4 - q^5 - q^6 + 2*q^7 - 2*q^8 - 2*q^9 + O(q^10)
sage: CuspForms(Gamma0(29), 2).0.qexp(10)
q - q^4 - q^5 - q^6 + 2*q^7 - 2*q^8 - 2*q^9 + O(q^10)
sage: F2 == CuspForms(Gamma0(29), 2).0
True
\end{verbatim}

The name $E_k_i$ stands for the $i$-th basis element of the Eisenstein subspace of weight $k$:

\begin{verbatim}
 sage: F4 = QM.from_polynomial(E4_0)
sage: F4.qexp(30)
1 + 240*q^29 + O(q^30)
sage: EisensteinForms(Gamma0(29), 4).0.qexp(30)
1 + 240*q^29 + O(q^30)
sage: F4 == EisensteinForms(Gamma0(29), 4).0
True
\end{verbatim}

One may also choose the name of the variables:

\begin{verbatim}
 sage: QM = QuasiModularForms(1)
sage: QM.polynomial_ring(names="P, Q, R")
Multivariate Polynomial Ring in P, Q, R over Rational Field
\end{verbatim}

\section{Graded quasimodular forms ring}

\subsection{Quasimodular forms of weight $(weight)$}

Return the space of quasimodular forms on this group of the given weight.

\textbf{INPUT:}

\begin{itemize}
\item weight (int, Integer)
\end{itemize}

\textbf{OUTPUT:} A quasimodular forms space of the given weight.

\textbf{EXAMPLES:}

\begin{verbatim}
 sage: QuasiModularForms(1).quasimodular_forms_of_weight(4)
Traceback (most recent call last):
...  
NotImplementedError: spaces of quasimodular forms of fixed weight not yet implemented
\end{verbatim}

\subsection{Some elements()}

Return a list of generators of \texttt{self}.

\textbf{EXAMPLES:}

\begin{verbatim}
 sage: QuasiModularForms(1).some_elements()
[1 - 24*q - 72*q^2 - 96*q^3 - 168*q^4 - 144*q^5 + O(q^6),
 1 + 240*q + 2160*q^2 + 6720*q^3 + 17520*q^4 + 30240*q^5 + O(q^6),
 1 - 504*q - 16632*q^2 - 122976*q^3 - 532728*q^4 - 1575504*q^5 + O(q^6)]
\end{verbatim}
weight_2_eisenstein_series()

Return the weight 2 Eisenstein series.

EXAMPLES:

```sage
QM = QuasiModularForms(1)
E2 = QM.weight_2_eisenstein_series(); E2
1 - 24*q - 72*q^2 - 96*q^3 - 168*q^4 - 144*q^5 + O(q^6)
E2.parent()
Ring of Quasimodular Forms for Modular Group SL(2,Z) over Rational Field
```

zero()

Return the zero element of this ring.

EXAMPLES:

```sage
QM = QuasiModularForms(1)
QM.zero()
0
QM.zero().is_zero()
True
```

3.2 Elements of quasimodular forms rings

AUTHORS:

- DAVID AYOTTE (2021-03-18): initial version

class sage.modular.quasimodform.element.QuasiModularFormsElement(parent, polynomial)

Bases: ModuleElement

A quasimodular forms ring element. Such an element is described by SageMath as a polynomial

\[ f_0 + f_1 E_2 + f_2 E_2^2 + \cdots + f_m E_2^m \]

where each \( f_i \) a graded modular form element (see GradedModularFormElement)

EXAMPLES:

```sage
QM = QuasiModularForms(1)
QM.gens()
[1 - 24*q - 72*q^2 - 96*q^3 - 168*q^4 - 144*q^5 + O(q^6),
1 + 240*q + 2160*q^2 + 6720*q^3 + 17520*q^4 + 30240*q^5 + O(q^6),
1 - 504*q - 16632*q^2 - 122976*q^3 - 532728*q^4 - 1575504*q^5 + O(q^6)]
QM.0 + QM.1
2 + 216*q + 2088*q^2 + 6624*q^3 + 17352*q^4 + 30996*q^5 + O(q^6)
QM.0 * QM.1
1 - 216*q + 3672*q^2 - 62496*q^3 - 322488*q^4 - 1121904*q^5 + O(q^6)
(QM.0)^2
1 - 48*q + 432*q^2 + 3264*q^3 + 9456*q^4 + 21600*q^5 + O(q^6)
QM.0 == QM.1
False
```

Quasimodular forms ring element can be created via a polynomial in \( E_2 \) over the ring of modular forms:
```python
sage: E2 = QM.polygen()
sage: E2.parent()
Univariate Polynomial Ring in E2 over Ring of Modular Forms for Modular Group SL(2, \mathbb{Z}) over Rational Field
sage: QM(E2)
1 - 24*q - 72*q^2 - 96*q^3 - 168*q^4 - 144*q^5 + O(q^6)
sage: M = QM.modular_forms_subring()
sage: QM(M.0 * E2 + M.1 * E2^2)
2 - 336*q + 4320*q^2 + 398400*q^3 - 3772992*q^4 - 89283168*q^5 + O(q^6)
```

One may convert a quasimodular form into a multivariate polynomial in the generators of the ring by calling `polynomial()`:

```python
sage: QM = QuasiModularForms(1)
sage: F = QM.0^2 + QM.1^2 + QM.0*QM.1*QM.2
sage: F.polynomial()
E2*E4*E6 + E4^2 + E2^2
```

If the group is not the full modular group, the default names of the generators are given by $E_{k,i}$ and $S_{k,i}$ to denote the $i$-th basis element of the weight $k$ Eisenstein subspace and cuspidal subspace respectively (for more details, see the documentation of `polynomial_ring()`)

```python
sage: QM = QuasiModularForms(Gamma1(4))
sage: F = (QM.0^4)*(QM.1^3) + QM.3
sage: F.polynomial()
-512*E2^4*E2_1^3 + E2^4*E3_0^2 + 48*E2^4*E3_1^2 + E3_0
```

### degree()

Return the weight of the given quasimodular form.

Note that the given form must be homogeneous. An alias of this method is `degree`.

**EXAMPLES:**

```python
sage: QM = QuasiModularForms(1)
sage: (QM.0).weight()
2
sage: (QM.0 * QM.1 + QM.2).weight()
6
sage: QM(1/2).weight()
0
sage: (QM.0).degree()
2
sage: (QM.0 + QM.1).weight()
Traceback (most recent call last):
...
ValueError: the given graded quasiform is not an homogeneous element
```

### derivative()

Return the derivative $\frac{d}{dq}$ of the given quasimodular form.

If the form is not homogeneous, then this method sums the derivative of each homogeneous component.

**EXAMPLES:**

```python
```
```python
sage: QM = QuasiModularForms(1)
sage: E2, E4, E6 = QM.gens()
sage: dE2 = E2.derivative(); dE2
-24*q - 144*q^2 - 288*q^3 - 672*q^4 - 720*q^5 + O(q^6)
sage: dE2 == (E2^2 - E4)/12 # Ramanujan identity
True
sage: dE4 = E4.derivative(); dE4
240*q + 4320*q^2 + 20160*q^3 + 70080*q^4 + 151200*q^5 + O(q^6)
sage: dE4 == (E2 * E4 - E6)/3 # Ramanujan identity
True
sage: dE6 = E6.derivative(); dE6
-504*q - 33264*q^2 - 368928*q^3 - 2130912*q^4 - 7877520*q^5 + O(q^6)
sage: dE6 == (E2 * E6 - E4^2)/2 # Ramanujan identity
True
```

Note that the derivative of a modular form is not necessarily a modular form:

```python
sage: dE4.is_modular_form()
False
sage: dE4.weight()
6
```

**homogeneous_component**(weight)

Return the homogeneous component of the given quasimodular form ring element.

An alias of this method is `homogeneous_component`.

EXAMPLES:

```python
sage: QM = QuasiModularForms(1)
sage: F = E2 + E4*E6 + E2^3*E6
sage: F[2]
1 - 24*q - 72*q^2 - 96*q^3 - 168*q^4 - 144*q^5 + O(q^6)
sage: F[10]
1 - 264*q - 135432*q^2 - 368928*q^3 - 2130912*q^4 - 7877520*q^5 + O(q^6)
sage: F[12]
1 - 576*q + 21168*q^2 + 308736*q^3 - 15034608*q^4 - 39208320*q^5 + O(q^6)
sage: F[4]
0
sage: F.homogeneous_component(2)
1 - 24*q - 72*q^2 - 96*q^3 - 168*q^4 - 144*q^5 + O(q^6)
```

**homogeneous_components()**

Return a dictionary where the values are the homogeneous components of the given graded form and the keys are the weights of those components.

EXAMPLES:

```python
sage: QM = QuasiModularForms(1)
sage: (QM.0).homogeneous_components()
{(2: 1 - 24*q - 72*q^2 - 96*q^3 - 168*q^4 - 144*q^5 + O(q^6))}
sage: (QM.0 + QM.1 + QM.2).homogeneous_components()
{(2: 1 - 24*q - 72*q^2 - 96*q^3 - 168*q^4 - 144*q^5 + O(q^6)),
 (4: 0)}
```

(continues on next page)
is_graded_modular_form()

Return whether the given quasimodular form is a graded modular form element (see \texttt{GradedModularFormElement}).

EXAMPLES:

```sage
sage: QM = QuasiModularForms(1)
sage: (QM.0).is_graded_modular_form()
False
sage: (QM.1).is_graded_modular_form()
True
sage: (QM.1 + QM.0^2).is_graded_modular_form()
False
sage: (QM.1**2 + QM.2).is_graded_modular_form()
True
sage: QM = QuasiModularForms(Gamma0(6))
sage: (QM.0).is_graded_modular_form()
False
sage: (QM.1 + QM.2 + QM.1 * QM.3).is_graded_modular_form()
True
sage: QM.zero().is_graded_modular_form()
True
sage: QM = QuasiModularForms(Gamma0(6))
sage: (QM.0).is_graded_modular_form()
False
sage: (QM.0 + QM.1*QM.2 + QM.3).is_graded_modular_form()
False
sage: (QM.1*QM.2 + QM.3).is_graded_modular_form()
True
```

**Note:** A graded modular form in SageMath is not necessarily a modular form as it can have mixed weight.
components. To check for modular forms only, see the method \texttt{is\_modular\_form()}.

\section*{is\_homogeneous()}

Return whether the graded quasimodular form is a homogeneous element, that is, it lives in a unique graded components of the parent of \texttt{self}.

\begin{verbatim}
EXAMPLES:
sage: QM = QuasiModularForms(1)
sage: E2, E4, E6 = QM.gens()
sage: (E2).is_homogeneous() True
sage: (E2 + E4).is_homogeneous() False
sage: (E2 * E4 + E6).is_homogeneous() True
sage: QM(1).is_homogeneous() True
sage: (1 + E2).is_homogeneous() False
sage: F = E6^3 + E4^4*E2 + (E4^2*E6)*E2^2 + (E4^3 + E6^2)*E2^3
sage: F.is_homogeneous() True
\end{verbatim}

\section*{is\_modular\_form()}

Return whether the given quasimodular form is a modular form.

\begin{verbatim}
EXAMPLES:
sage: QM = QuasiModularForms(1)
sage: (QM.0).is_modular_form() False
sage: (QM.1).is_modular_form() True
sage: (QM.1 + QM.2).is_modular_form() # mixed weight components False
sage: QM.zero().is_modular_form() True
sage: QM = QuasiModularForms(Gamma0(4))
sage: (QM.0).is_modular_form() False
sage: (QM.1).is_modular_form() True
\end{verbatim}

\section*{is\_one()}

Return whether the given quasimodular form is 1, i.e. the multiplicative identity.

\begin{verbatim}
EXAMPLES:
sage: QM = QuasiModularForms(1)
sage: QM.one().is_one() True
sage: QM(1).is_one() True
\end{verbatim}
**is_zero()**

Return whether the given quasimodular form is zero.

**EXAMPLES:**

```python
sage: QM = QuasiModularForms(1)
sage: QM.zero().is_zero()
True
sage: QM(0).is_zero()
True
sage: QM(1/2).is_zero()
False
sage: (QM.0).is_zero()
False
sage: QM = QuasiModularForms(Gamma0(2))
sage: QM(0).is_zero()
True
```

**polynomial(names=None)**

Return a multivariate polynomial such that every variable corresponds to a generator of the ring, ordered by the method: `gens()`.

An alias of this method is `to_polynomial`.

**INPUT:**

- `names` (str, default: `None`) – a list or tuple of names (strings), or a comma separated string. Defines the names for the generators of the multivariate polynomial ring. The default names are of the form ABCk where k is a number corresponding to the weight of the form ABC.

**OUTPUT:** A multivariate polynomial in the variables names

**EXAMPLES:**

```python
sage: QM = QuasiModularForms(1)
sage: (QM.0 + QM.1).polynomial()
E4 + E2
sage: (1/2 + QM.0 + 2*QM.1^2 + QM.0*QM.2).polynomial()
E2*E6 + 2*E4^2 + E2 + 1/2
```

Check that github issue #34569 is fixed:

```python
sage: QM = QuasiModularForms(Gamma1(3))
sage: QM.ngens()
5
sage: (QM.0 + QM.1 + QM.2*QM.1 + QM.0*QM.2).polynomial()
E3_1*E4_0 + E2*E3_0 + E2 + E2_0
```

**q_expansion(prec=6)**

Return the $q$-expansion of the given quasimodular form up to precision `prec` (default: 6).
An alias of this method is \texttt{qexp}.

\begin{verbatim}
\texttt{sage: QM = QuasiModularForms()}
\texttt{sage: E2 = QM.0}
\texttt{sage: E2.q_expansion()}
1 - 24*q - 72*q^2 - 96*q^3 - 168*q^4 - 144*q^5 + O(q^6)
\texttt{sage: E2.q_expansion(prec=10)}
1 - 24*q - 72*q^2 - 96*q^3 - 168*q^4 - 144*q^5 - 288*q^6 - 192*q^7 - 360*q^8 - ...
\rightarrow 312*q^9 + O(q^10)
\end{verbatim}

\texttt{qexp}(\texttt{prec=}6)

Return the \(q\)-expansion of the given quasimodular form up to precision \texttt{prec} (default: 6).

An alias of this method is \texttt{qexp}.

\begin{verbatim}
\texttt{sage: QM = QuasiModularForms()}
\texttt{sage: E2 = QM.0}
\texttt{sage: E2.q_expansion()}
1 - 24*q - 72*q^2 - 96*q^3 - 168*q^4 - 144*q^5 + O(q^6)
\texttt{sage: E2.q_expansion(prec=10)}
1 - 24*q - 72*q^2 - 96*q^3 - 168*q^4 - 144*q^5 - 288*q^6 - 192*q^7 - 360*q^8 - ...
\rightarrow 312*q^9 + O(q^10)
\end{verbatim}

\texttt{serre_derivative}()

Return the Serre derivative of the given quasimodular form.

If the form is not homogeneous, then this method sums the Serre derivative of each homogeneous component.

\begin{verbatim}
\texttt{sage: QM = QuasiModularForms(1)}
\texttt{sage: E2, E4, E6 = QM.gens()}
\texttt{sage: DE2 = E2.serre_derivative(); DE2}
-1/6 - 16*q - 216*q^2 - 832*q^3 - 2248*q^4 - 4320*q^5 + O(q^6)
\texttt{sage: DE2 == (-E2^2 - E4)/12}  
True
\texttt{sage: DE4 = E4.serre_derivative(); DE4}
-1/3 + 168*q + 5544*q^2 + 40992*q^3 + 177576*q^4 + 525168*q^5 + O(q^6)
\texttt{sage: DE4 == (-1/3) * E6}  
True
\texttt{sage: DE6 = E6.serre_derivative(); DE6}
-1/2 - 240*q - 30960*q^2 - 525120*q^3 - 3963120*q^4 - 18750240*q^5 + O(q^6)
\texttt{sage: DE6 == (-1/2) * E4^2}  
True
\end{verbatim}

The Serre derivative raises the weight of homogeneous elements by 2:

\begin{verbatim}
\texttt{sage: F = E6 + E4 * E2}
\texttt{sage: F.weight()}  
6
\texttt{sage: F.serre_derivative().weight()}  
8
\end{verbatim}
Check that github issue #34569 is fixed:

```
sage: QM = QuasiModularForms(Gamma1(3))
sage: E2 = QM.weight_2_eisenstein_series()
sage: E2.serre_derivative()
-1/6 - 16\cdot q - 216\cdot q^2 - 832\cdot q^3 - 2248\cdot q^4 - 4320\cdot q^5 + O(q^6)
sage: F = QM.0 + QM.1*QM.2
```

**to_polynomial**(names=None)

Return a multivariate polynomial such that every variable corresponds to a generator of the ring, ordered by the method: `gens()`.

An alias of this method is `to_polynomial`.

**INPUT:**

- `names` (str, default: None) – a list or tuple of names (strings), or a comma separated string. Defines the names for the generators of the multivariate polynomial ring. The default names are of the form \(ABCk\) where \(k\) is a number corresponding to the weight of the form \(ABC\).

**OUTPUT:** A multivariate polynomial in the variables `names`

**EXAMPLES:**

```
sage: QM = QuasiModularForms(1)
sage: (QM.0 + QM.1).polynomial()
E4 + E2
sage: (1/2 + QM.0 + 2*QM.1*QM.2).polynomial()
E2*E6 + 2*E4^2 + E2 + 1/2
```

Check that github issue #34569 is fixed:

```
sage: QM = QuasiModularForms(Gamma1(3))
sage: QM.ngens()
5
sage: (QM.0 + QM.1 + QM.2*QM.1 + QM.3*QM.4).polynomial()
E3_1*E4_0 + E2_0*E3_0 + E2 + E2_0
```

**weight()**

Return the weight of the given quasimodular form.

Note that the given form must be homogeneous. An alias of this method is `degree`.

**EXAMPLES:**

```
sage: QM = QuasiModularForms(1)
sage: (QM.0).weight()
2
sage: (QM.0 + QM.1 + QM.2).weight()
6
sage: QM(1/2).weight()
0
sage: (QM.0).degree()
2
sage: (QM.0 + QM.1).weight()
Traceback (most recent call last):
  ... ValueError: the given graded quasiform is not an homogeneous element
```
weights_list()

Return the list of the weights of all the graded components of the given graded quasimodular form.

EXAMPLES:

```
sage: QM = QuasiModularForms(1)
sage: (QM.0).weights_list()
[2]
sage: (QM.0 + QM.1 + QM.2).weights_list()
[2, 4, 6]
sage: (QM.0 * QM.1 + QM.2).weights_list()
[6]
sage: QM(1/2).weights_list()
[0]
sage: QM = QuasiModularForms(Gamma1(3))
sage: (QM.0 + QM.1 + QM.2*QM.1 + QM.3*QM.4).weights_list()
[2, 5, 7]
```
4.1 Dirichlet characters

A `DirichletCharacter` is the extension of a homomorphism

\[(\mathbb{Z}/N\mathbb{Z})^* \to R^*,\]

for some ring $R$, to the map $\mathbb{Z}/N\mathbb{Z} \to R$ obtained by sending those $x \in \mathbb{Z}/N\mathbb{Z}$ with $\gcd(N, x) > 1$ to 0.

**EXAMPLES:**

```
sage: G = DirichletGroup(35)
sage: x = G.gens()
sage: e = x[0]*x[1]^2; e
Dirichlet character modulo 35 of conductor 35 mapping 22 |--> zeta12^3, 31 |--> zeta12^2
              _-  1
sage: e.order()
12
```

This illustrates a canonical coercion:

```
sage: e = DirichletGroup(5, QQ).0
sage: f = DirichletGroup(5,CyclotomicField(4)).0
sage: e*f
Dirichlet character modulo 5 of conductor 5 mapping 2 |--> -zeta4
```

**AUTHORS:**

- William Stein (2005-09-02): Fixed bug in comparison of Dirichlet characters. It was checking that their values were the same, but not checking that they had the same level!
- William Stein (2006-01-07): added more examples
- William Stein (2006-05-21): added examples of everything; fix a lot of tiny bugs and design problem that became clear when creating examples.
- Craig Citro (2008-02-16): speed up `__call__` method for Dirichlet characters, miscellaneous fixes
- Julian Rueth (2014-03-06): use UniqueFactory to cache DirichletGroups

```
class sage.modular.dirichlet.DirichletCharacter(parent, x, check=True)
    Bases: MultiplicativeGroupElement
```

A Dirichlet character.
bar()

Return the complex conjugate of this Dirichlet character.

EXAMPLES:

```python
sage: e = DirichletGroup(5).0
sage: e
Dirichlet character modulo 5 of conductor 5 mapping 2 |--> zeta4
sage: e.bar()
Dirichlet character modulo 5 of conductor 5 mapping 2 |--> -zeta4
```

base_ring()

Return the base ring of this Dirichlet character.

EXAMPLES:

```python
sage: G = DirichletGroup(11)
sage: G.gen(0).base_ring()
Cyclotomic Field of order 10 and degree 4
sage: G = DirichletGroup(11, RationalField())
sage: G.gen(0).base_ring()
Rational Field
```

bernoulli(k, algorithm='recurrence', cache=True, **opts)

Return the generalized Bernoulli number \( B_{k,\epsilon} \).

INPUT:

- \( k \) – a non-negative integer
- \( \text{algorithm} \) – either 'recurrence' (default) or 'definition'
- \( \text{cache} \) – if True, cache answers
- \( **\text{opts} \) – optional arguments; not used directly, but passed to the \( \text{bernoulli}() \) function if this is called

OUTPUT:

Let \( \epsilon \) be a (not necessarily primitive) character of modulus \( N \). This function returns the generalized Bernoulli number \( B_{k,\epsilon} \), as defined by the following identity of power series (see for example [DI1995], Section 2.2):

\[
\sum_{a=1}^{N} \frac{\epsilon(a)te^{at}}{e^{Nt} - 1} = \sum_{k=0}^{\infty} \frac{B_{k,\epsilon} t^k}{k!}.
\]

ALGORITHM:

The 'recurrence' algorithm computes generalized Bernoulli numbers via classical Bernoulli numbers using the formula in [Coh2007], Proposition 9.4.5; this is usually optimal. The definition algorithm uses the definition directly.

**Warning:** In the case of the trivial Dirichlet character modulo 1, this function returns \( B_{1,\epsilon} = 1/2 \), in accordance with the above definition, but in contrast to the value \( B_{1} = -1/2 \) for the classical Bernoulli number. Some authors use an alternative definition giving \( B_{1,\epsilon} = -1/2 \); see the discussion in [Coh2007], Section 9.4.1.

EXAMPLES:
```python
sage: G = DirichletGroup(13)
sage: e = G.0
sage: e.bernoulli(5)
7430/13*zeta12^3 - 34750/13*zeta12^2 - 11380/13*zeta12 + 9110/13
sage: eps = DirichletGroup(9).0
sage: eps.bernoulli(3)
10*zeta6 + 4
sage: eps.bernoulli(3, algorithm="definition")
10*zeta6 + 4
```

**change_ring** (*R*)

Return the base extension of *self* to *R*.

**INPUT:**

- *R* – either a ring admitting a conversion map from the base ring of *self*, or a ring homomorphism with the base ring of *self* as its domain

**EXAMPLES:**

```python
sage: e = DirichletGroup(7, QQ).0
sage: f = e.change_ring(QuadraticField(3, 'a'))
sage: f.parent()
Group of Dirichlet characters modulo 7 with values in Number Field in a with defining polynomial x^2 - 3 with a = 1.732050807568878?
```

```python
sage: e = DirichletGroup(13).0
sage: e.change_ring(QQ)
Traceback (most recent call last):
...
TypeError: Unable to coerce zeta12 to a rational
```

We test the case where *R* is a map (github issue #18072):

```python
sage: K.<i> = QuadraticField(-1)
sage: chi = DirichletGroup(5, K)[1]
sage: chi(2)
i
sage: f = K.complex_embeddings()[0]
sage: psi = chi.change_ring(f)
sage: psi(2)
-1.83697019872103e-16 - 1.00000000000000*I
```

**conductor()**

Compute and return the conductor of this character.

**EXAMPLES:**

```python
sage: G.<a,b> = DirichletGroup(20)
sage: a.conductor()
4
sage: b.conductor()
5
sage: (a*b).conductor()
20
```

### 4.1. Dirichlet characters

297
conrey_number()

Return the Conrey number for this character.

This is a positive integer coprime to \( q \) that identifies a Dirichlet character of modulus \( q \).

See https://www.lmfdb.org/knowledge/show/character.dirichlet.conrey

EXAMPLES:

```sage
chi4 = DirichletGroup(4).gen()
sage: chi4.conrey_number()
3
chi = DirichletGroup(24)([1,-1,-1]); chi
Dirichlet character modulo 24 of conductor 24
mapping 7 |--> 1, 13 |--> -1, 17 |--> -1
sage: chi.conrey_number()
5

chi = DirichletGroup(60)([1,-1,I])
sage: chi.conrey_number()
17

chi = DirichletGroup(420)([1,-1,-I,1])
sage: chi.conrey_number()
113
```

decomposition()

Return the decomposition of \( \text{self} \) as a product of Dirichlet characters of prime power modulus, where the prime powers exactly divide the modulus of this character.

EXAMPLES:

```sage
G.<a,b> = DirichletGroup(20)
sage: c = a*b
decomposition(); d
[Dirichlet character modulo 4 of conductor 4 mapping 3 |--> -1, Dirichlet character modulo 5 of conductor 5 mapping 2 |--> zeta4]
sage: d[0].parent()
Group of Dirichlet characters modulo 4 with values in Cyclotomic Field of order 4 and degree 2
sage: d[1].parent()
Group of Dirichlet characters modulo 5 with values in Cyclotomic Field of order 4 and degree 2
```

We cannot multiply directly, since coercion of one element into the other parent fails in both cases:

```sage
d[0]*d[1] == c
Traceback (most recent call last):
...
TypeError: unsupported operand parent(s) for *: 'Group of Dirichlet characters modulo 4 with values in Cyclotomic Field of order 4 and degree 2' and 'Group of Dirichlet characters modulo 5 with values in Cyclotomic Field of order 4 and degree 2'
```

We can multiply if we are explicit about where we want the multiplication to take place.
Conductors that are divisible by various powers of 2 present some problems as the multiplicative group modulo \(2^k\) is trivial for \(k = 1\) and non-cyclic for \(k \geq 3\):

\[
\text{sage: } (\text{DirichletGroup}(18).0).\text{decomposition()}
\]
\[
[\text{Dirichlet character modulo 2 of conductor 1, Dirichlet character modulo 9 of}_{\text{conductor 9 mapping 2 } \rightarrow \text{zeta6}}]
\]

\[
\text{sage: } (\text{DirichletGroup}(36).0).\text{decomposition()}
\]
\[
[\text{Dirichlet character modulo 4 of conductor 4 mapping 3 } \rightarrow \text{-1, Dirichlet}_\text{character modulo 9 of conductor 1 mapping 2 } \rightarrow \text{1}]
\]

\[
\text{sage: } (\text{DirichletGroup}(72).0).\text{decomposition()}
\]
\[
[\text{Dirichlet character modulo 8 of conductor 4 mapping 7 } \rightarrow \text{-1, 5 } \rightarrow \text{1,}_{\text{Dirichlet character modulo 9 of conductor 1 mapping 2 } \rightarrow \text{1}}]
\]

\section{element()}
Return the underlying \(\mathbb{Z}/n\mathbb{Z}\)-module vector of exponents.

\section{extend(M)}
Return the extension of this character to a Dirichlet character modulo the multiple \(M\) of the modulus.

\section{fixed_field()}
Given a Dirichlet character, this will return the abelian extension fixed by the kernel of the corresponding Galois character.
Given a Dirichlet character, this will return a polynomial generating the abelian extension fixed by the kernel of the corresponding Galois character.

ALGORITHM: (Sage)

A formula by Gauss for the products of periods; see Disquisitiones §343. See the source code for more.

OUTPUT:

• a polynomial with integer coefficients

EXAMPLES:

```python
sage: G = DirichletGroup(37)
sage: chi = G.0
sage: psi = chi^18
sage: psi.fixed_field_polynomial()
x^2 + x - 9

sage: G = DirichletGroup(7)
sage: chi = G.0^2
sage: chi
Dirichlet character modulo 7 of conductor 7 mapping 3 |--> zeta6 - 1
sage: chi.fixed_field_polynomial()
x^3 + x^2 - 2*x - 1

sage: G = DirichletGroup(31)
sage: chi = G.0^n
sage: chi^6
Dirichlet character modulo 31 of conductor 31 mapping 3 |--> zeta30^6
sage: psi = chi^6
sage: psi.fixed_field_polynomial()
x^5 + x^4 - 12*x^3 - 21*x^2 + x + 5
```
\[
x^5 + x^4 - 12x^3 - 21x^2 + x + 5
\]

```sage
G = DirichletGroup(7)
sage: chi = G.0
sage: chi.fixed_field_polynomial()
x^6 + x^5 + x^4 + x^3 + x^2 + x + 1
```

```sage
G = DirichletGroup(1001)
sage: chi = G.0
sage: psi = chi^3
sage: psi.order()
2
sage: psi.fixed_field_polynomial(algorithm="pari")
x^2 + x + 2
```

With the Sage implementation:

```sage
G = DirichletGroup(37)
sage: chi = G.0
sage: psi = chi^18
sage: psi.fixed_field_polynomial(algorithm="sage")
x^2 + x - 9
```

```sage
G = DirichletGroup(7)
sage: chi = G.0^2
sage: chi
Dirichlet character modulo 7 of conductor 7 mapping 3 |--> zeta6 - 1
sage: chi.fixed_field_polynomial(algorithm="sage")
x^3 + x^2 - 2*x - 1
```

```sage
G = DirichletGroup(31)
sage: chi = G.0
sage: chi^6
Dirichlet character modulo 31 of conductor 31 mapping 3 |--> zeta30^6
sage: psi = chi^6
sage: psi.fixed_field_polynomial(algorithm="sage")
x^5 + x^4 - 12x^3 - 21x^2 + x + 5
```

```sage
G = DirichletGroup(7)
sage: chi = G.0
sage: chi.fixed_field_polynomial(algorithm="sage")
x^6 + x^5 + x^4 + x^3 + x^2 + x + 1
```

```sage
G = DirichletGroup(1001)
sage: chi = G.0
sage: psi = chi^3
sage: psi.order()
2
sage: psi.fixed_field_polynomial(algorithm="sage")
x^2 + x + 2
```

The algorithm must be one of \texttt{sage} or \texttt{pari}:
sage: G = DirichletGroup(1001)
sage: chi = G.0
sage: psi = chi^3
galois_orbit(sort=True)
Return the orbit of this character under the action of the absolute Galois group of the prime subfield of the base ring.

EXAMPLES:

sage: G = DirichletGroup(30); e = G.1
sage: e.galois_orbit()
[Dirichlet character modulo 30 of conductor 5 mapping 11 |--> 1, 7 |--> -zeta4,
  Dirichlet character modulo 30 of conductor 5 mapping 11 |--> 1, 7 |--> zeta4]

Another example:

sage: G = DirichletGroup(13)
sage: G.galois_orbits()
[[Dirichlet character modulo 13 of conductor 1 mapping 2 |--> 1],
  ...,
  [Dirichlet character modulo 13 of conductor 13 mapping 2 |--> -1]]
sage: e = G.0
sage: e
Dirichlet character modulo 13 of conductor 13 mapping 2 |--> zeta12
sage: e.galois_orbit()
[Dirichlet character modulo 13 of conductor 13 mapping 2 |--> zeta12,
  Dirichlet character modulo 13 of conductor 13 mapping 2 |--> -zeta12^2 + 1]

A non-example:

sage: chi = DirichletGroup(7, Integers(9), zeta = Integers(9)(2)).0
sage: chi.galois_orbit()
Traceback (most recent call last):
...
TypeError: Galois orbits only defined if base ring is an integral domain

gauss_sum(a=1)
Return a Gauss sum associated to this Dirichlet character.

The Gauss sum associated to $\chi$ is

$$g_a(\chi) = \sum_{r \in \mathbb{Z}/m\mathbb{Z}} \chi(r) \zeta^{ar},$$

where $m$ is the modulus of $\chi$ and $\zeta$ is a primitive $m^{th}$ root of unity.

FACTS: If the modulus is a prime $p$ and the character is nontrivial, then the Gauss sum has absolute value $\sqrt{p}$.

CACHING: Computed Gauss sums are not cached with this character.

EXAMPLES:

```python
sage: G = DirichletGroup(3)
sage: e = G([-1])
sage: e.gauss_sum(1)
2*zeta6 - 1
sage: e.gauss_sum(2)
-2*zeta6 + 1
sage: norm(e.gauss_sum())
3
```

```python
sage: G = DirichletGroup(13)
sage: e = G.0
sage: e.gauss_sum()
-zeta156^46 + zeta156^45 + zeta156^42 + zeta156^41 + 2*zeta156^40 + zeta156^37 -
   zeta156^36 - zeta156^34 - zeta156^33 - zeta156^31 + 2*zeta156^30 + zeta156^28 -
   zeta156^24 - zeta156^22 + zeta156^21 + zeta156^20 - zeta156^19 + zeta156^18 -
   zeta156^16 + zeta156^15 - 2*zeta156^14 - zeta156^10 + zeta156^8 +
   -zeta156^7 + zeta156^6 + zeta156^5 - zeta156^4 - zeta156^2 - 1
sage: factor(norm(e.gauss_sum()))
13^24
```

See also:

- `sage.arith.misc.gauss_sum()` for general finite fields
- `sage.rings.padics.misc.gauss_sum()` for a $p$-adic version

`gauss_sum_numerical(prec=53, a=1)`

Return a Gauss sum associated to this Dirichlet character as an approximate complex number with `prec` bits of precision.

INPUT:

- `prec` – integer (default: 53), bits of precision
- `a` – integer, as for `gauss_sum()`.

The Gauss sum associated to $\chi$ is

$$g_a(\chi) = \sum_{r \in \mathbb{Z}/m\mathbb{Z}} \chi(r) \zeta^{ar},$$

where $m$ is the modulus of $\chi$ and $\zeta$ is a primitive $m^{th}$ root of unity.
EXAMPLES:

```python
sage: G = DirichletGroup(3)
sage: e = G.0
sage: abs(e.gauss_sum_numerical())
1.7320508075...
sage: sqrt(3.0)
1.73205080756888
sage: e.gauss_sum_numerical(a=2)
-.07497205... - 1.7320508075...*I
sage: e.gauss_sum_numerical(a=2, prec=100)
4.733165431326078324703713917e-30 - 1.732050807568877293527463415*I
sage: G = DirichletGroup(13)
sage: H = DirichletGroup(13, CC)
sage: e = G.0
sage: f = H.0
sage: e.gauss_sum_numerical()
-3.07497205... + 1.8826966926...*I
sage: f.gauss_sum_numerical()
-3.07497205... + 1.8826966926...*I
sage: abs(e.gauss_sum_numerical())
3.60555127546...
sage: abs(f.gauss_sum_numerical())
3.60555127546...
sage: sqrt(13.0)
3.60555127546399
```

`is_even()`

Return True if and only if $\varepsilon(-1) = 1$.

EXAMPLES:

```python
sage: G = DirichletGroup(13)
sage: e = G.0
sage: e.is_even()
False
sage: e(-1)
-1
sage: [e.is_even() for e in G]
[True, False, True, False, True, False, True, False, True, False, True, False]
```

```python
sage: G = DirichletGroup(13, CC)
sage: e = G.0
sage: e.is_even()
False
sage: e(-1)
-1.000000...
```

```python
sage: [e.is_even() for e in G]
[True, False, True, False, True, False, True, False, True, False, True, False]
```

```python
sage: G = DirichletGroup(100000, CC)
sage: G.1.is_even()
True
```

Note that `is_even` need not be the negation of `is_odd`, e.g., in characteristic 2:
Modular Forms, Release 10.0

```python
sage: G.<e> = DirichletGroup(13, GF(4,'a'))
sage: e.is_even()
True
sage: e.is_odd()
True
```

**is_odd()**

Return True if and only if \( \varepsilon(-1) = -1 \).

EXAMPLES:

```python
sage: G = DirichletGroup(13)
sage: e = G.0
sage: e.is_odd()
True
sage: [e.is_odd() for e in G]
[False, True, False, True, False, True, False, True, False, True, False, True]
```

```python
sage: G = DirichletGroup(100000, CC)
sage: G.0.is_odd()
True
```

Note that `is_even` need not be the negation of `is_odd`, e.g., in characteristic 2:

```python
sage: G.<e> = DirichletGroup(13, GF(4,'a'))
sage: e.is_even()
True
sage: e.is_odd()
True
```

**is_primitive()**

Return True if and only if this character is primitive, i.e., its conductor equals its modulus.

EXAMPLES:

```python
sage: G.<a,b> = DirichletGroup(20)
sage: a.is_primitive()
False
sage: b.is_primitive()
False
sage: (a*b).is_primitive()
True
sage: G.<a,b> = DirichletGroup(20, CC)
sage: a.is_primitive()
False
sage: b.is_primitive()
False
```

(continues on next page)
(continued from previous page)

```
sage: (a*b).is_primitive()
True
```

**is_trivial()**

Return True if this is the trivial character, i.e., has order 1.

**EXAMPLES:**

```
sage: G.<a,b> = DirichletGroup(20)
sage: a.is_trivial()
False
sage: (a^2).is_trivial()
True
```

**jacobi_sum(char, check=True)**

Return the Jacobi sum associated to these Dirichlet characters (i.e., $J(self, char)$).

This is defined as

$$J(\chi, \psi) = \sum_{a \in \mathbb{Z}/N\mathbb{Z}} \chi(a)\psi(1-a)$$

where $\chi$ and $\psi$ are both characters modulo $N$.

**EXAMPLES:**

```
sage: D = DirichletGroup(13)
sage: e = D.0
sage: f = D[-2]
sage: e.jacobi_sum(f)
3*zeta12^2 + 2*zeta12 - 3
sage: f.jacobi_sum(e)
3*zeta12^2 + 2*zeta12 - 3
sage: p = 7
sage: DP = DirichletGroup(p)
sage: f = DP.0
sage: e.jacobi_sum(f)
Traceback (most recent call last):
... NotImplementedError: Characters must be from the same Dirichlet Group.
sage: all_jacobi_sums = [(DP[i].values_on_gens(),DP[j].values_on_gens(),DP[i].˓→jacobi_sum(DP[j])) for i in range(p-1) for j in range(i, p-1)]
sage: for s in all_jacobi_sums:
... print(s)
((1,), (1,), 5)
((1,), (zeta6,), -1)
((1,), (zeta6 - 1,), -1)
((1,), (-1,), -1)
((1,), (-zeta6,), -1)
((1,), (-zeta6 + 1,), -1)
(((zeta6,), (zeta6,), -zeta6 + 3)
(((zeta6,), (zeta6 - 1,), 2*zeta6 + 1)
```

(continues on next page)
Let's check that trivial sums are being calculated correctly:

```
sage: N = 13
sage: D = DirichletGroup(N)

sage: g = D(1)
sage: g.jacobi_sum(g)
11
sage: sum([g(x)*g(1-x) for x in IntegerModRing(N)])
11
```

And sums where exactly one character is nontrivial (see github issue #6393):

```
sage: G = DirichletGroup(5); X = G.list(); Y=X[0]; Z=X[1]
sage: Y.jacobi_sum(Z)
-1
sage: Z.jacobi_sum(Y)
-1
```

Now let's take a look at a non-prime modulus:

```
sage: N = 9
sage: D = DirichletGroup(N)
sage: g = D(1)
sage: g.jacobi_sum(g)
3
```

We consider a sum with values in a finite field:

```
sage: g = DirichletGroup(17, GF(9,'a')).0
sage: g.jacobi_sum(g**2)
2*a
```

**kernel()**

Return the kernel of this character.

**OUTPUT:** Currently the kernel is returned as a list. This may change.

**EXAMPLES:**

---

4.1. Dirichlet characters

307
```python
sage: G.<a,b> = DirichletGroup(20)
sage: a.kernel()
[1, 9, 13, 17]
sage: b.kernel()
[1, 11]
```

**kloosterman_sum**\((a=1, b=0)\)

Return the “twisted” Kloosterman sum associated to this Dirichlet character.

This includes Gauss sums, classical Kloosterman sums, Salié sums, etc.

The Kloosterman sum associated to \(\chi\) and the integers \(a,b\) is

\[
K(a, b, \chi) = \sum_{r \in (\mathbb{Z}/m\mathbb{Z})^\times} \chi(r) \zeta^{ar + br^{-1}},
\]

where \(m\) is the modulus of \(\chi\) and \(\zeta\) is a primitive \(m\) th root of unity. This reduces to the Gauss sum if \(b = 0\).

This method performs an exact calculation and returns an element of a suitable cyclotomic field; see also `kloosterman_sum_numerical()`, which gives an inexact answer (but is generally much quicker).

CACHING: Computed Kloosterman sums are not cached with this character.

**EXAMPLES:**

```python
sage: G = DirichletGroup(3)
sage: e = G([-1])
sage: e.kloosterman_sum(3,5)
-2*zeta6 + 1
sage: G = DirichletGroup(20)
sage: e = G([1 for u in G.unit_gens()])
sage: e.kloosterman_sum(7,17)
-2*zeta20^6 + 2*zeta20^4 + 4
```

**kloosterman_sum_numerical**\((prec=53, a=1, b=0)\)

Return the Kloosterman sum associated to this Dirichlet character as an approximate complex number with \(prec\) bits of precision.

See also `kloosterman_sum()`, which calculates the sum exactly (which is generally slower).

**INPUT:**

- \(prec\) – integer (default: 53), bits of precision
- \(a\) – integer, as for `kloosterman_sum()`
- \(b\) – integer, as for `kloosterman_sum()`.

**EXAMPLES:**

```python
sage: G = DirichletGroup(3)
sage: e = G.0
The real component of the numerical value of e is near zero:

```python
sage: v = e.kloosterman_sum_numerical()
sage: v.real() < 1.0e15
True
sage: v.imag()
```
1.73205080756888
sage: G = DirichletGroup(20)
sage: e = G.1
sage: e.kloosterman_sum_numerical(53,3,11)
3.80422606518061 - 3.80422606518061*I

**level()**

Synonym for modulus.

EXAMPLES:

```
sage: e = DirichletGroup(100, QQ).0
sage: e.level()
100
```

**lfunction**(prec=53, algorithm='pari')

Return the L-function of self.

The result is a wrapper around a PARI L-function or around the lcalc program.

INPUT:

- `prec` – precision (default 53)
- `algorithm` – ‘pari’ (default) or ‘lcalc’

EXAMPLES:

```
sage: G.<a,b> = DirichletGroup(20)
sage: L = a.lfunction(); L
PARI L-function associated to Dirichlet character modulo 20
of conductor 4 mapping 11 |--> -1, 17 |--> 1
sage: L(4)
0.988944551741105
```

With the algorithm “lcalc”:

```
sage: a = a.primitive_character()
sage: L = a.lfunction(algorithm='lcalc'); L
L-function with complex Dirichlet coefficients
sage: L.value(4)  # abs tol 1e-8
0.988944551741105 + 0.*I
```

**lmfdb_page()**

Open the LMFDB web page of the character in a browser.

See https://www.lmfdb.org

EXAMPLES:

```
sage: E = DirichletGroup(4).gen()
sage: E.lmfdb_page()  # optional -- webbrowser
```

**maximize_base_ring()**

Let

\[ \varepsilon : (\mathbb{Z}/N\mathbb{Z})^* \to \mathbb{Q}(\zeta_n) \]
be a Dirichlet character. This function returns an equal Dirichlet character
\[ \chi : (\mathbb{Z}/N\mathbb{Z})^* \rightarrow \mathbb{Q}(\zeta_m) \]

where \( m \) is the least common multiple of \( n \) and the exponent of \( (\mathbb{Z}/N\mathbb{Z})^* \).

**EXAMPLES:**

```python
sage: G.<a,b> = DirichletGroup(20,QQ)
sage: b.maximize_base_ring()
Dirichlet character modulo 20 of conductor 5 mapping 11 |--> 1, 17 |--> -1
sage: b.maximize_base_ring().base_ring()
Cyclotomic Field of order 4 and degree 2
sage: DirichletGroup(20).base_ring()
Cyclotomic Field of order 4 and degree 2
```

**minimize_base_ring()**

Return a Dirichlet character that equals this one, but over as small a subfield (or subring) of the base ring as possible.

**Note:** This function is currently only implemented when the base ring is a number field. It is the identity function in characteristic \( p \).

**EXAMPLES:**

```python
sage: G = DirichletGroup(13)
sage: e = DirichletGroup(13).0
sage: e.base_ring()
Cyclotomic Field of order 12 and degree 4
sage: e.minimize_base_ring().base_ring()
Cyclotomic Field of order 12 and degree 4
sage: (e^2).minimize_base_ring().base_ring()
Cyclotomic Field of order 6 and degree 2
sage: (e^3).minimize_base_ring().base_ring()
Cyclotomic Field of order 4 and degree 2
sage: (e^12).minimize_base_ring().base_ring()
Rational Field
```

**modulus()**

Return the modulus of this character.

**EXAMPLES:**

```python
sage: e = DirichletGroup(100, QQ).0
sage: e.modulus()
100
sage: e.conductor()
4
```

**multiplicative_order()**

Return the order of this character.

**EXAMPLES:**
sage: e = DirichletGroup(100).1
sage: e.order()  # same as multiplicative_order, since group is multiplicative
20
sage: e.multiplicative_order()
20
sage: e = DirichletGroup(100).0
sage: e.multiplicative_order()
2

primitive_character()

Return the primitive character associated to self.

EXAMPLES:

sage: e = DirichletGroup(100).0; e
Dirichlet character modulo 100 of conductor 4 mapping 51 |--> -1, 77 |--> 1
sage: e.conductor()
4
sage: f = e.primitive_character(); f
Dirichlet character modulo 4 of conductor 4 mapping 3 |--> -1
sage: f.modulus()
4

restrict(M)

Return the restriction of this character to a Dirichlet character modulo the divisor M of the modulus, which must also be a multiple of the conductor of this character.

EXAMPLES:

sage: e = DirichletGroup(100).0
sage: e.modulus()
100
sage: e.conductor()
4
sage: e.restrict(20)
Dirichlet character modulo 20 of conductor 4 mapping 11 |--> -1, 17 |--> 1
sage: e.restrict(4)
Dirichlet character modulo 4 of conductor 4 mapping 3 |--> -1
sage: e.restrict(50)
Traceback (most recent call last):
... ValueError: conductor(=4) must divide M(=50)

values()

Return a list of the values of this character on each integer between 0 and the modulus.

EXAMPLES:

sage: e = DirichletGroup(20)(1)

(continues on next page)
\begin{verbatim}
\textbf{sage:} e = DirichletGroup(20).gen(1)
\textbf{sage:} e.values()
\[0, 1, 0, -\zeta_4, 0, 0, \zeta_4, 0, -1, 0, 1, 0, -\zeta_4, 0, 0, \zeta_4, 0, -1\]
\textbf{sage:} e = DirichletGroup(21).gen(0); e.values()
\[0, 1, -1, 0, 1, -1, 0, 0, -1, 0, 1, 0, -1, 0, 1, -1, 0, 1, -1\]
\textbf{sage:} e = DirichletGroup(21, base_ring=GF(37)).gen(0); e.values()
\[0, 1, 36, 0, 1, 36, 0, 0, 36, 0, 1, 36, 0, 1, 0, 0, 1, 36, 0, 1, 36\]
\textbf{sage:} e = DirichletGroup(21, base_ring=GF(3)).gen(0); e.values()
\[0, 1, 2, 0, 1, 2, 0, 0, 2, 0, 1, 2, 0, 1, 0, 1, 2, 0, 1, 2\]

\textbf{sage:} chi = DirichletGroup(100151, CyclotomicField(10)).0
\textbf{sage:} ls = chi.values(); ls[0:10]
\[0, 1, -\zeta_{10}^3, -\zeta_{10}, -\zeta_{10}, 1, \zeta_{10}^3 - \zeta_{10}^2 + \zeta_{10} - 1, \zeta_{10}, \zeta_{10}^3 - \zeta_{10}^2 + \zeta_{10} - 1, \zeta_{10}^2\]
\end{verbatim}

\textbf{values_on_gens()}

Return a tuple of the values of self on the standard generators of \((\mathbb{Z}/N\mathbb{Z})^*\), where \(N\) is the modulus.

\textbf{EXAMPLES:}

\begin{verbatim}
\textbf{sage:} e = DirichletGroup(16)([-1, 1])
\textbf{sage:} e.values_on_gens()
\((-1, 1)\)
\end{verbatim}

\textbf{Note:} The constructor of \texttt{DirichletCharacter} sets the cache of \texttt{element()} or of \texttt{values_on_gens()}. The cache of one of these methods needs to be set for the other method to work properly, these caches have to be stored when pickling an instance of \texttt{DirichletCharacter}.

\textbf{class} \texttt{sage.modular.dirichlet.DirichletGroupFactory}

\textbf{Bases:} \texttt{UniqueFactory}

Construct a group of Dirichlet characters modulo \(N\).

\textbf{INPUT:}

- \(N\) – positive integer
- \texttt{base\_ring} – commutative ring; the value ring for the characters in this group (default: the cyclotomic field \(\mathbb{Q}(\zeta_n)\), where \(n\) is the exponent of \((\mathbb{Z}/N\mathbb{Z})^*\))
- \texttt{zeta} – (optional) root of unity in base\_ring
- \texttt{zeta\_order} – (optional) positive integer; this must be the order of zeta if both are specified
- \texttt{names} – ignored (needed so \texttt{G.<...> = DirichletGroup(...)} notation works)
• **integral** – boolean (default: **False**); whether to replace the default cyclotomic field by its rings of integers as the base ring. This is ignored if **base_ring** is not **None**.

**OUTPUT:**

The group of Dirichlet characters modulo $N$ with values in a subgroup $V$ of the multiplicative group $R^*$ of **base_ring**. This is the group of homomorphisms $(\mathbb{Z}/N\mathbb{Z})^* \rightarrow V$ with pointwise multiplication. The group $V$ is determined as follows:

- If both **zeta** and **zeta_order** are omitted, then $V$ is taken to be $R^*$, or equivalently its $n$-torsion subgroup, where $n$ is the exponent of $(\mathbb{Z}/N\mathbb{Z})^*$. Many operations, such as finding a set of generators for the group, are only implemented if $V$ is cyclic and a generator for $V$ can be found.

- If **zeta** is specified, then $V$ is taken to be the cyclic subgroup of $R^*$ generated by **zeta**. If **zeta_order** is also given, it must be the multiplicative order of **zeta**; this is useful if the base ring is not exact or if the order of **zeta** is very large.

- If **zeta** is not specified but **zeta_order** is, then $V$ is taken to be the group of roots of unity of order dividing **zeta_order** in $R$. In this case, $R$ must be a domain (so $V$ is cyclic), and $V$ must have order **zeta_order**. Furthermore, a generator **zeta** of $V$ is computed, and an error is raised if such **zeta** cannot be found.

**EXAMPLES:**

The default base ring is a cyclotomic field of order the exponent of $(\mathbb{Z}/N\mathbb{Z})^*$:

```
sage: DirichletGroup(20)
Group of Dirichlet characters modulo 20 with values in Cyclotomic Field of order 4 and degree 2
```

We create the group of Dirichlet character mod 20 with values in the rational numbers:

```
sage: G = DirichletGroup(20, QQ); G
Group of Dirichlet characters modulo 20 with values in Rational Field
sage: G.order()
4
sage: G.base_ring()
Rational Field
```

The elements of $G$ print as lists giving the values of the character on the generators of $(\mathbb{Z}/N\mathbb{Z})^*$:

```
sage: list(G)
[Dirichlet character modulo 20 of conductor 1 mapping 11 |--> 1, 17 |--> 1,
  Dirichlet character modulo 20 of conductor 4 mapping 11 |--> -1, 17 |--> 1,
  Dirichlet character modulo 20 of conductor 5 mapping 11 |--> 1, 17 |--> -1,
  Dirichlet character modulo 20 of conductor 20 mapping 11 |--> -1, 17 |--> -1]
```

Next we construct the group of Dirichlet character mod 20, but with values in $\mathbb{Q}(\zeta_n)$:

```
sage: G = DirichletGroup(20)
sage: G.1
Dirichlet character modulo 20 of conductor 5 mapping 11 |--> 1, 17 |--> zeta4
```

We next compute several invariants of $G$:

```
sage: G.gens()
(Dirichlet character modulo 20 of conductor 4 mapping 11 |--> -1, 17 |--> 1,
  Dirichlet character modulo 20 of conductor 5 mapping 11 |--> 1, 17 |--> zeta4)
```
In this example we create a Dirichlet group with values in a number field:

```python
sage: R.<x> = PolynomialRing(QQ)
sage: K.<a> = NumberField(x^4 + 1)
sage: DirichletGroup(5, K)
Group of Dirichlet characters modulo 5 with values in Number Field in a with defining polynomial x^4 + 1
```

An example where we give \( \zeta \), but not its order:

```python
sage: G = DirichletGroup(5, K, a); G
Group of Dirichlet characters modulo 5 with values in the group of order 8 generated by a in Number Field in a with defining polynomial x^4 + 1
sage: G.list()
[Dirichlet character modulo 5 of conductor 1 mapping 2 |---> 1, Dirichlet character modulo 5 of conductor 5 mapping 2 |---> a^2, Dirichlet character modulo 5 of conductor 5 mapping 2 |---> -1, Dirichlet character modulo 5 of conductor 5 mapping 2 |---> -a^2]
```

We can also restrict the order of the characters, either with or without specifying a root of unity:

```python
sage: DirichletGroup(5, K, zeta=-1, zeta_order=2)
Group of Dirichlet characters modulo 5 with values in the group of order 2 generated by -1 in Number Field in a with defining polynomial x^4 + 1
sage: DirichletGroup(5, K, zeta_order=2)
Group of Dirichlet characters modulo 5 with values in the group of order 2 generated by -1 in Number Field in a with defining polynomial x^4 + 1
```

We compute a Dirichlet group over a large prime field:

```python
sage: p = next_prime(10^40)
sage: g = DirichletGroup(19, GF(p)); g
Group of Dirichlet characters modulo 19 with values in Finite Field of size 100000000000000000000000000000000000000121
```

Note that the root of unity has small order, i.e., it is not the largest order root of unity in the field:
sage: g.zeta_order()
2

sage: r4 = CyclotomicField(4).ring_of_integers()
sage: G = DirichletGroup(60, r4)
sage: G.gens()
(Dirichlet character modulo 60 of conductor 4 mapping 31 |--> -1, 41 |--> 1, 37 |--> -1, Dirichlet character modulo 60 of conductor 3 mapping 31 |--> 1, 41 |--> -1, 37 |--> 1, Dirichlet character modulo 60 of conductor 5 mapping 31 |--> 1, 41 |--> -1, 37 |--> zeta4)
sage: val = G.gens()[2].values_on_gens()[2] ; val
zeta4
sage: parent(val)
Gaussian Integers in Cyclotomic Field of order 4 and degree 2
sage: r4.residue_field(r4.ideal(29).factor()[0][0])(val)
doctest:warning ... DeprecationWarning: ...
17
sage: r4.residue_field(r4.ideal(29).factor()[0][0])(val) * GF(29)(3)
22
sage: r4.residue_field(r4.ideal(29).factor()[0][0])(G.gens()[2].values_on_gens()[2]) * 3
22
sage: parent(r4.residue_field(r4.ideal(29).factor()[0][0])(G.gens()[2].values_on_gens()[2]) * 3)
Residue field of Fractional ideal (-2*zeta4 + 5)

sage: DirichletGroup(60, integral=True)
Group of Dirichlet characters modulo 60 with values in Gaussian Integers in Cyclotomic Field of order 4 and degree 2
sage: parent(DirichletGroup(60, integral=True).gens()[2].values_on_gens()[2])
Gaussian Integers in Cyclotomic Field of order 4 and degree 2

If the order of zeta cannot be determined automatically, we can specify it using zeta_order:

sage: DirichletGroup(7, CC, zeta=exp(2*pi*I/6))
Traceback (most recent call last):
  ... NotImplementedError: order of element not known
sage: DirichletGroup(7, CC, zeta=exp(2*pi*I/6), zeta_order=6)
Group of Dirichlet characters modulo 7 with values in the group of order 6 generated by 0.500000000000000 + 0.866025403784439*I in Complex Field with 53 bits of precision

If the base ring is not a domain (in which case the group of roots of unity is not necessarily cyclic), some operations still work, such as creation of elements:

sage: G = DirichletGroup(5, Zmod(15)); G
Group of Dirichlet characters modulo 5 with values in Ring of integers modulo 15
sage: chi = G[[13]]; chi
Dirichlet character modulo 5 of conductor 5 mapping 2 |--> 13
sage: chi^2
Dirichlet character modulo 5 of conductor 5 mapping 2 |--> 4

(continues on next page)
Other operations only work if `zeta` is specified:

```
sage: G.gens()
Traceback (most recent call last):
... 
NotImplementedError: factorization of polynomials over rings with composite
→ characteristic is not implemented
sage: G = DirichletGroup(5, Zmod(15), zeta=2); G
Group of Dirichlet characters modulo 5 with values in the group of order 4,
→ generated by 2 in Ring of integers modulo 15
sage: G.gens()
(Dirichlet character modulo 5 of conductor 5 mapping 2 |--> 2,)
```

```python
def create_key(N, base_ring=None, zeta=None, zeta_order=None, names=None, integral=False):
    """Create a key that uniquely determines a Dirichlet group."""

def create_object(version, key, **extra_args):
    """Create the object from the key (extra arguments are ignored).
    This is only called if the object was not found in the cache."""

class DirichletGroup_class(base_ring, modulus, zeta, zeta_order):
    """Group of Dirichlet characters modulo \( N \) with values in a ring \( R \)."""

    Element
       alias of DirichletCharacter

    base_extend(R)
       Return the base extension of self to \( R \).

       INPUT:

       * \( R \) – either a ring admitting a coercion map from the base ring of self, or a ring homomorphism with the base ring of self as its domain

       EXAMPLES:

```
sage: G = DirichletGroup(7,QQ); G
Group of Dirichlet characters modulo 7 with values in Rational Field
sage: H = G.base_extend(CyclotomicField(6)); H
Group of Dirichlet characters modulo 7 with values in Cyclotomic Field of order 6 and degree 2
```

Note that the root of unity can change:

```
sage: H.zeta()
zeta6
```

This method (in contrast to `change_ring()`) requires a coercion map to exist:
sage: G.base_extend(ZZ)
Traceback (most recent call last):
...  
TypeError: no coercion map from Rational Field to Integer Ring is defined

Base-extended Dirichlet groups do not silently get roots of unity with smaller order than expected (github issue #6018):

```
sage: G = DirichletGroup(10, QQ).base_extend(CyclotomicField(4))
sage: H = DirichletGroup(10, CyclotomicField(4))
sage: G is H
True
sage: G3 = DirichletGroup(31, CyclotomicField(3))
sage: G5 = DirichletGroup(31, CyclotomicField(5))
sage: K30 = CyclotomicField(30)
sage: G3.gen(0).base_extend(K30) * G5.gen(0).base_extend(K30)
Dirichlet character modulo 31 of conductor 31 mapping 3 |--> -zeta30^7 + zeta30^5 + zeta30^4 + zeta30^3 - zeta30 - 1
```

When a root of unity is specified, base extension still works if the new base ring is not an integral domain:

```
sage: f = DirichletGroup(17, ZZ, zeta=-1).0
sage: g = f.base_extend(Integers(15))
sage: g(3)
14
sage: g.parent().zeta()
14
```

```
change_ring(R, zeta=None, zeta_order=None)
Return the base extension of self to R.

INPUT:

• R – either a ring admitting a conversion map from the base ring of self, or a ring homomorphism with the base ring of self as its domain

• zeta – (optional) root of unity in R

• zeta_order – (optional) order of zeta

EXAMPLES:

```
sage: G = DirichletGroup(7,QQ); G
Group of Dirichlet characters modulo 7 with values in Rational Field
sage: G.change_ring(CyclotomicField(6))
Group of Dirichlet characters modulo 7 with values in Cyclotomic Field of order
˓→ 6 and degree 2
```

```
decomposition()
Return the Dirichlet groups of prime power modulus corresponding to primes dividing modulus.
(Note that if the modulus is 2 mod 4, there will be a “factor” of (Z/2Z)*, which is the trivial group.)

EXAMPLES:

```
4.1. Dirichlet characters
317
sage: DirichletGroup(20).decomposition()
[
Group of Dirichlet characters modulo 4 with values in Cyclotomic Field of order 4 and degree 2,
Group of Dirichlet characters modulo 5 with values in Cyclotomic Field of order 4 and degree 2
]
sage: DirichletGroup(20,GF(5)).decomposition()
[
Group of Dirichlet characters modulo 4 with values in Finite Field of size 5,
Group of Dirichlet characters modulo 5 with values in Finite Field of size 5
]

exponent()

Return the exponent of this group.

EXAMPLES:

sage: DirichletGroup(20).exponent()
4
sage: DirichletGroup(20,GF(3)).exponent()
2
sage: DirichletGroup(20,GF(2)).exponent()
1
sage: DirichletGroup(37).exponent()
36

galois_orbits(v=None, reps_only=False, sort=True, check=True)

Return a list of the Galois orbits of Dirichlet characters in self, or in v if v is not None.

INPUT:

• v - (optional) list of elements of self
• reps_only - (optional: default False) if True only returns representatives for the orbits.
• sort - (optional: default True) whether to sort the list of orbits and the orbits themselves (slightly faster if False).
• check - (optional, default: True) whether or not to explicitly coerce each element of v into self.

The Galois group is the absolute Galois group of the prime subfield of Frac(R). If R is not a domain, an error will be raised.

EXAMPLES:

sage: DirichletGroup(20).galois_orbits()
[Dirichlet character modulo 20 of conductor 20 mapping 11 |--> -1, 17 |--> -1], ...
[Dirichlet character modulo 20 of conductor 1 mapping 11 |--> 1, 17 |--> 1]

sage: DirichletGroup(17, Integers(6), zeta=Integers(6)(5)).galois_orbits()
Traceback (most recent call last):
...
TypeError: Galois orbits only defined if base ring is an integral domain

(continues on next page)
Modular Forms, Release 10.0

sage: DirichletGroup(17, Integers(9), zeta=Integers(9)(2)).galois_orbits()
Traceback (most recent call last):
  ... 
TypeError: Galois orbits only defined if base ring is an integral domain

```python
sage: G = DirichletGroup(20)
sage: G.gen(0)
Dirichlet character modulo 20 of conductor 4 mapping 11 |--> -1, 17 |--> 1
sage: G. gen(1)
Dirichlet character modulo 20 of conductor 5 mapping 11 |--> 1, 17 |--> zeta4
sage: G.gen(2)
Traceback (most recent call last):
  ... 
IndexError: n(=2) must be between 0 and 1
```

```python
sage: G. gen(-1)
Traceback (most recent call last):
  ... 
IndexError: n=-1) must be between 0 and 1
```

```python
sage: G.gens()
(Dirichlet character modulo 20 of conductor 4 mapping 11 |--> -1, 17 |--> 1,
  Dirichlet character modulo 20 of conductor 5 mapping 11 |--> 1, 17 |--> zeta4)
```

```python
sage: G.integers_mod()
Ring of integers modulo 20
```

```python
sage: DirichletGroup(5).list()
[Dirichlet character modulo 5 of conductor 1 mapping 2 |---> 1,
  Dirichlet character modulo 5 of conductor 5 mapping 2 |---> zeta4,
  Dirichlet character modulo 5 of conductor 5 mapping 2 |---> -1,
  Dirichlet character modulo 5 of conductor 5 mapping 2 |---> -zeta4]
```

4.1. Dirichlet characters
modulus()  
Return the modulus of self.

EXAMPLES:

```
sage: G = DirichletGroup(20)
sage: G.modulus()
20
```

gens()  
Return the number of generators of self.

EXAMPLES:

```
sage: G = DirichletGroup(20)
sage: G.ngens()
2
```

order()  
Return the number of elements of self.

This is the same as len(self).

EXAMPLES:

```
sage: DirichletGroup(20).order()
8
sage: DirichletGroup(37).order()
36
```

random_element()  
Return a random element of self.

The element is computed by multiplying a random power of each generator together, where the power is between 0 and the order of the generator minus 1, inclusive.

EXAMPLES:

```
sage: D = DirichletGroup(37)
sage: g = D.random_element()
sage: g.parent() is D
True
sage: g**36
Dirichlet character modulo 37 of conductor 1 mapping 2 |--> 1
sage: S = set(D.random_element().conductor() for _ in range(100))
sage: while S != {1, 37}:
    S.add(D.random_element().conductor())
sage: DirichletGroup(20)
sage: g = D.random_element()
sage: g.parent() is D
True
sage: g**4
Dirichlet character modulo 20 of conductor 1 mapping 11 |--> 1, 17 |--> 1
sage: S = set(D.random_element().conductor() for _ in range(100))
sage: while S != {1, 4, 5, 20}:
    S.add(D.random_element().conductor())
```
S.add(D.random_element().conductor())

sage: D = DirichletGroup(60)
sage: g = D.random_element()
sage: g.parent() is D
True
sage: g**4
Dirichlet character modulo 60 of conductor 1 mapping 31 |---> 1, 41 |---> 1, 37 |---> 1
sage: S = set(D.random_element().conductor() for _ in range(100))
sage: while S != {1, 3, 5, 12, 15, 20, 60}:
    ....:     S.add(D.random_element().conductor())

unit_gens()
Return the minimal generators for the units of \((\mathbb{Z}/N\mathbb{Z})^*\), where \(N\) is the modulus of self.

EXAMPLES:

sage: DirichletGroup(37).unit_gens()
(2,)
sage: DirichletGroup(20).unit_gens()
(11, 17)
sage: DirichletGroup(60).unit_gens()
(31, 41, 37)
sage: DirichletGroup(20, QQ).unit_gens()
(11, 17)

zeta()
Return the chosen root of unity in the base ring.

EXAMPLES:

sage: DirichletGroup(37).zeta()
zeta36
sage: DirichletGroup(20).zeta()
zeta4
sage: DirichletGroup(60).zeta()
zeta4
sage: DirichletGroup(60, QQ).zeta()
-1
sage: DirichletGroup(60, GF(25, 'a')).zeta()
2

zeta_order()
Return the order of the chosen root of unity in the base ring.

EXAMPLES:

sage: DirichletGroup(20).zeta_order()
4
sage: DirichletGroup(60).zeta_order()
4
sage: DirichletGroup(60, GF(25, 'a')).zeta_order()
sage: DirichletGroup(19).zeta_order()
18

sage.modular.dirichlet.TrivialCharacter(N, base_ring=Rational Field)

Return the trivial character of the given modulus, with values in the given base ring.

EXAMPLES:

sage: t = trivial_character(7)
sage: [t(x) for x in [0..20]]
[0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1]
sage: t(1).parent()
Rational Field
sage: trivial_character(7, Integers(3))(1).parent()
Ring of integers modulo 3

sage.modular.dirichlet.is_DirichletCharacter(x)

Return True if x is of type DirichletCharacter.

EXAMPLES:

sage: from sage.modular.dirichlet import is_DirichletCharacter
sage: is_DirichletCharacter(trivial_character(3))
True
sage: is_DirichletCharacter([1])
False

sage.modular.dirichlet.is_DirichletGroup(x)

Return True if x is a Dirichlet group.

EXAMPLES:

sage: from sage.modular.dirichlet import is_DirichletGroup
sage: is_DirichletGroup(DirichletGroup(11))
True
sage: is_DirichletGroup(11)
False
sage: is_DirichletGroup(DirichletGroup(11).0)
False

sage.modular.dirichlet.kronecker_character(d)

Return the quadratic Dirichlet character (d/.) of minimal conductor.

EXAMPLES:

sage: kronecker_character(97*389*997^2)
Dirichlet character modulo 37733 of conductor 37733 mapping 1557 |--> -1, 37346 |--> -1
sage: a = kronecker_character(1)
sage: b = DirichletGroup(2401,QQ)(a)    # NOTE -- over QQ!
sage: b.modulus()
2401
sage.modular.dirichlet.kronecker_character_upside_down(d)
Return the quadratic Dirichlet character (.d) of conductor d, for d0.

EXAMPLES:

```python
sage: kronecker_character_upside_down(97*389*997^2)
Dirichlet character modulo 37506941597 of conductor 37733 mapping 13533432536 |--> -
    \rightarrow 1, 22369178537 |--> -1, 14266017175 |--> 1
```

AUTHORS:
• Jon Hanke (2006-08-06)

sage.modular.dirichlet.trivial_character(N, base_ring=Rational Field)
Return the trivial character of the given modulus, with values in the given base ring.

EXAMPLES:

```python
sage: t = trivial_character(7)
sage: [t(x) for x in [0..20]]
[0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1]
sage: t(1).parent()
Rational Field
sage: trivial_character(7, Integers(3))(1).parent()
Ring of integers modulo 3
```

### 4.2 The set $\mathbb{P}^1(Q)$ of cusps

EXAMPLES:

```python
sage: Cusps
Set $\mathbb{P}^1(QQ)$ of all cusps
sage: Cusp(oo)
Infinity
```

class sage.modular.cusps.Cusp(a, b=None, parent=None, check=True)
Bases: Element
A cusp.
A cusp is either a rational number or infinity, i.e., an element of the projective line over Q. A Cusp is stored as a pair (a,b), where gcd(a,b)=1 and a,b are of type Integer.

EXAMPLES:

```python
sage: a = Cusp(2/3); b = Cusp(oo)
sage: a.parent()
Set $\mathbb{P}^1(QQ)$ of all cusps
sage: a.parent() is b.parent()
True
```
apply($g$)
Return $g(self)$, where $g=[a,b,c,d]$ is a list of length 4, which we view as a linear fractional transformation.

EXAMPLES: Apply the identity matrix:

```
sage: Cusp(0).apply([1,0,0,1])
0
sage: Cusp(0).apply([0,-1,1,0])
Infinity
sage: Cusp(0).apply([1,-3,0,1])
-3
```

denominator()
Return the denominator of the cusp $a/b$.

EXAMPLES:

```
sage: x = Cusp(6,9); x
2/3
sage: x.denominator()
3
sage: Cusp(oo).denominator()
0
sage: Cusp(-5/10).denominator()
2
```
galois_action($t$, $N$)
Suppose this cusp is $\alpha$, $G$ a congruence subgroup of level $N$ and $\sigma$ is the automorphism in the Galois group of $\mathbb{Q}(\zeta_N)/\mathbb{Q}$ that sends $\zeta_N$ to $\zeta_N^t$. Then this function computes a cusp $\beta$ such that $\sigma([\alpha]) = [\beta]$, where $[\alpha]$ is the equivalence class of $\alpha$ modulo $G$.

This code only needs as input the level and not the group since the action of Galois for a congruence group $G$ of level $N$ is compatible with the action of the full congruence group $\Gamma(N)$.

INPUT:

• $t$ – integer that is coprime to $N$
• $N$ – positive integer (level)

OUTPUT:

• a cusp

**Warning:** In some cases $N$ must fit in a long long, i.e., there are cases where this algorithm isn’t fully implemented.

**Note:** Modular curves can have multiple non-isomorphic models over $\mathbb{Q}$. The action of Galois depends on such a model. The model over $\mathbb{Q}$ of $X(G)$ used here is the model where the function field $\mathbb{Q}(X(G))$ is given by the functions whose Fourier expansion at $\infty$ have their coefficients in $\mathbb{Q}$. For $X(N) := X(\Gamma(N))$ the corresponding moduli interpretation over $\mathbb{Z}[1/N]$ is that $X(N)$ parametrizes pairs $(E, a)$ where $E$ is a (generalized) elliptic curve and $a : \mathbb{Z}/N\mathbb{Z} \times \mu_N \to E$ is a closed immersion such that the Weil pairing of $a(1,1)$ and $a(0, \zeta_N)$ is $\zeta_N$. In this parameterisation the point $z \in H$ corresponds to the pair $(E_z, a_z)$ with $E_z = C/(z\mathbb{Z} + \mathbb{Z})$ and $a_z : \mathbb{Z}/N\mathbb{Z} \times \mu_N \to E$ given by $a_z(1,1) = z/N$ and $a_z(0, \zeta_N) = 1/N$. Similarly $X_1(N) := X(\Gamma_1(N))$ parametrizes pairs $(E, a)$ where $a : \mu_N \to E$ is a closed immersion.
EXAMPLES:

```python
sage: Cusp(1/10).galois_action(3, 50)
1/170
sage: Cusp(oo).galois_action(3, 50)
Infinity
sage: c = Cusp(0).galois_action(3, 50); c
50/17
sage: Gamma0(50).reduce_cusp(c)
0
```

Here we compute the permutations of the action for \( t=3 \) on cusps for \( \Gamma_0(50) \).

```python
sage: N = 50; t=3; G = Gamma0(N); C = G.cusps()
sage: cl = lambda z: exists(C, lambda y: y.is_gamma0_equiv(z, N))[1]
sage: for i in range(5):
    ....:       print((i, t^i))
    ....:       print([cl(alpha.galois_action(t^i,N)) for alpha in C])
(0, 1)
[0, 1/25, 1/10, 1/5, 3/10, 2/5, 1/2, 3/5, 7/10, 4/5, 9/10, Infinity]
(1, 3)
[0, 1/25, 7/10, 2/5, 1/10, 4/5, 1/2, 1/5, 9/10, 3/5, 3/10, Infinity]
(2, 9)
[0, 1/25, 9/10, 4/5, 7/10, 3/5, 1/2, 2/5, 3/10, 1/5, 1/10, Infinity]
(3, 27)
[0, 1/25, 3/10, 3/5, 9/10, 1/5, 1/2, 4/5, 1/10, 2/5, 7/10, Infinity]
(4, 81)
[0, 1/25, 1/10, 1/5, 3/10, 2/5, 1/2, 3/5, 7/10, 4/5, 9/10, Infinity]
```

REFERENCES:

- Section 1.3 of Glenn Stevens, “Arithmetic on Modular Curves”
- There is a long comment about our algorithm in the source code for this function.

AUTHORS:

- William Stein, 2009-04-18

`is_gamma0_equiv(other, N, transformation=None)`

Return whether self and other are equivalent modulo the action of \( \Gamma_0(N) \) via linear fractional transformations.

INPUT:

- `other` - Cusp
- `N` - an integer (specifies the group \( \Gamma_0(N) \))
- `transformation` - None (default) or either the string ‘matrix’ or ‘corner’. If ‘matrix’, it also returns a matrix in \( \Gamma_0(N) \) that sends self to other. The matrix is chosen such that the lower left entry is as small as possible in absolute value. If ‘corner’ (or True for backwards compatibility), it returns only the upper left entry of such a matrix.

OUTPUT:

- a boolean - True if self and other are equivalent
- a matrix or an integer - returned only if transformation is ‘matrix’ or ‘corner’, respectively.

EXAMPLES:

4.2. The set \( \mathbb{P}^1(\mathbb{Q}) \) of cusps

is_gamma1_equiv(other, N)

Return whether self and other are equivalent modulo the action of Gamma_1(N) via linear fractional transformations.

INPUT:
• other - Cusp
• N - an integer (specifies the group Gamma_1(N))

OUTPUT:
• bool - True if self and other are equivalent
• int - 0, 1 or -1, gives further information about the equivalence: If the two cusps are u1/v1 and u2/v2, then they are equivalent if and only if v1 = v2 (mod N) and u1 = u2 (mod gcd(v1,N)) or v1 = -v2 (mod N) and u1 = -u2 (mod gcd(v1,N)) The sign is +1 for the first and -1 for the second. If the two cusps are not equivalent then 0 is returned.

EXAMPLES:

sage: x = Cusp(2,3)
sage: y = Cusp(4,5)
sage: x.is_gamma1_equiv(y,2)  
(True, 1)
sage: x.is_gamma1_equiv(y,3)  
(False, 0)
sage: z = Cusp(QQ(x) + 10)  
sage: x.is_gamma1_equiv(z,10)  
(True, 1)
sage: z = Cusp(1,0)
is_gamma_h_equiv(other, G)

Return a pair \((b, t)\), where \(b\) is True or False as self and other are equivalent under the action of \(G\), and \(t\) is 1 or -1, as described below.

Two cusps \(u_1/v_1\) and \(u_2/v_2\) are equivalent modulo \(\Gamma_H(N)\) if and only if \(v_1 = h \cdot v_2 \pmod{N}\) and \(u_1 = h^{(-1)} \cdot u_2 \pmod{\gcd(v_1, N)}\) or \(v_1 = -h \cdot v_2 \pmod{N}\) and \(u_1 = -h^{(-1)} \cdot u_2 \pmod{\gcd(v_1, N)}\) for some \(h \in H\). Then \(t\) is 1 or -1 as \(c\) and \(c'\) fall into the first or second case, respectively.

**INPUT:**
- other - Cusp
- G - a congruence subgroup \(\Gamma_H(N)\)

**OUTPUT:**
- bool - True if self and other are equivalent
- int - -1, 0, 1; extra info

**EXAMPLES:**

```python
sage: x = Cusp(2,3)
sage: y = Cusp(4,5)
sage: x.is_gamma_h_equiv(y, GammaH(13,[2]))
(True, 1)
sage: x.is_gamma_h_equiv(y, GammaH(13,[5]))
(False, 0)
sage: x.is_gamma_h_equiv(y, GammaH(5,[[]]))
(False, 0)
sage: x.is_gamma_h_equiv(y, GammaH(23,[4]))
(True, -1)
```

Enumerating the cusps for a space of modular symbols uses this function.

```python
sage: G = GammaH(25,[6]) ; M = G.modular_symbols() ; M
Modular Symbols space of dimension 11 for Congruence Subgroup Gamma_H(25) with \(--> H\) generated by \([6]\) of weight 2 with sign 0 over Rational Field
sage: M.cusps()
sage: len(M.cusps())
12
```

This is always one more than the associated space of weight 2 Eisenstein series.

```python
sage: G = GammaH(25,[6]); M = G.modular_symbols(); M
Modular Symbols space of dimension 11 for Congruence Subgroup Gamma_H(25) with \(--> H\) generated by \([6]\) of weight 2 with sign 0 over Rational Field
sage: M.cusps()
```

This is always one more than the associated space of weight 2 Eisenstein series.

```python
sage: G = GammaH(25,[6]); M = G.modular_symbols(); M
Modular Symbols space of dimension 11 for Congruence Subgroup Gamma_H(25) with \(--> H\) generated by \([6]\) of weight 2 with sign 0 over Rational Field
sage: M.cusps()
```
Modular Forms, Release 10.0

(continued from previous page)

\[ \Gamma_1 \text{ for Congruence Subgroup } \Gamma(25) \text{ with } \Gamma \text{ generated by } [6] \text{ of weight } 2 \]

\[ \text{with sign } 0 \text{ over Rational Field} \]

\[ \texttt{sage: G.dimension_cusp_forms}(2) \]

\[ \texttt{0} \]

\section*{is\_infinity()}

Returns True if this is the cusp infinity.

\textbf{EXAMPLES:}

\begin{verbatim}
\texttt{sage: Cusp(3/5).is\_infinity()} False
\texttt{sage: Cusp(1,0).is\_infinity()} True
\texttt{sage: Cusp(0,1).is\_infinity()} False
\end{verbatim}

\section*{numerator()}

Return the numerator of the cusp a/b.

\textbf{EXAMPLES:}

\begin{verbatim}
\texttt{sage: x = Cusp(6,9); x} 2/3
\texttt{sage: x.numerator()} 2
\texttt{sage: Cusp(oo).numerator()} 1
\texttt{sage: Cusp(-5/10).numerator()} -1
\end{verbatim}

\section*{class \texttt{sage.modular.cusps.Cusps\_class}}

\texttt{Bases: Singleton, Parent}

The set of cusps.

\textbf{EXAMPLES:}

\begin{verbatim}
\texttt{sage: C = Cusps; C} Set P^1(QQ) of all cusps
\texttt{sage: loads(C.dumps()) == C} True
\end{verbatim}

\textbf{Element}

alias of \texttt{Cusp}
4.3 Dimensions of spaces of modular forms

AUTHORS:

• William Stein

• Jordi Quer

ACKNOWLEDGEMENT: The dimension formulas and implementations in this module grew out of a program that Bruce Kaskel wrote (around 1996) in PARI, which Kevin Buzzard subsequently extended. I (William Stein) then implemented it in C++ for Hecke. I also implemented it in Magma. Also, the functions for dimensions of spaces with nontrivial character are based on a paper (that has no proofs) by Cohen and Oesterlé [CO1977]. The formulas for $\Gamma_H(N)$ were found and implemented by Jordi Quer.

The formulas here are more complete than in Hecke or Magma.

Currently the input to each function below is an integer and either a Dirichlet character $\varepsilon$ or a finite index subgroup of $\text{SL}_2(\mathbb{Z})$. If the input is a Dirichlet character $\varepsilon$, the dimensions are for subspaces of $M_k(\Gamma_1(N), \varepsilon)$, where $N$ is the modulus of $\varepsilon$.

These functions mostly call the methods dimension_cusp_forms, dimension_modular_forms and so on of the corresponding congruence subgroup classes.

REFERENCES:

sage.modular.dims.CO_delta(r, p, N, eps)

This is used as an intermediate value in computations related to the paper of Cohen-Oesterlé.

INPUT:

• r – positive integer

• p – a prime

• N – positive integer

• eps – character

OUTPUT: element of the base ring of the character

EXAMPLES:

```
sage: G.<eps> = DirichletGroup(7)
sage: sage.modular.dims.CO_delta(1,5,7,eps^3)
2
```

sage.modular.dims.CO_nu(r, p, N, eps)

This is used as an intermediate value in computations related to the paper of Cohen-Oesterlé.

INPUT:

• r – positive integer

• p – a prime

• N – positive integer

• eps – character

OUTPUT: element of the base ring of the character

EXAMPLES:
sage: G.<eps> = DirichletGroup(7)
sage: G.<eps> = DirichletGroup(7)
sage: sage.modular.dims.CO_nu(1,7,7,eps)
-1

sage.modular.dims.CohenOesterle(eps, k)
Compute the Cohen-Oesterlé function associate to eps, k.
This is a summand in the formula for the dimension of the space of cusp forms of weight 2 with character \( \varepsilon \).

INPUT:

• eps – Dirichlet character
• k – integer

OUTPUT: element of the base ring of eps.

EXAMPLES:

sage: G.<eps> = DirichletGroup(7)
sage: sage.modular.dims.CohenOesterle(eps, 2)
-2/3
sage: sage.modular.dims.CohenOesterle(eps, 4)
-1

sage.modular.dims.dimension_cusp_forms(X, k=2)
The dimension of the space of cusp forms for the given congruence subgroup or Dirichlet character.

INPUT:

• X – congruence subgroup or Dirichlet character or integer
• k – weight (integer)

EXAMPLES:

sage: from sage.modular.dims import dimension_cusp_forms
sage: dimension_cusp_forms(5,4)
1
sage: dimension_cusp_forms(Gamma0(11),2)
1
sage: dimension_cusp_forms(Gamma1(13),2)
2
sage: dimension_cusp_forms(DirichletGroup(13).0^2,2)
1
sage: dimension_cusp_forms(DirichletGroup(13).0,3)
1
sage: dimension_cusp_forms(Gamma0(11),2)
1
sage: dimension_cusp_forms(Gamma0(11),0)
0
sage: dimension_cusp_forms(Gamma0(1),12)
1
(continues on next page)
sage: dimension_cusp_forms(Gamma0(1),2)
0
sage: dimension_cusp_forms(Gamma0(1),4)
0
sage: dimension_cusp_forms(Gamma0(389),2)
32
sage: dimension_cusp_forms(Gamma0(389),4)
97
sage: dimension_cusp_forms(Gamma0(2005),2)
199
sage: dimension_cusp_forms(Gamma0(11),1)
0
sage: dimension_cusp_forms(Gamma1(11),2)
1
sage: dimension_cusp_forms(Gamma1(1),12)
1
sage: dimension_cusp_forms(Gamma1(1),2)
0
sage: dimension_cusp_forms(Gamma1(1),4)
0
sage: dimension_cusp_forms(Gamma1(389),2)
6112
sage: dimension_cusp_forms(Gamma1(389),4)
18721
sage: dimension_cusp_forms(Gamma1(2005),2)
159201
sage: dimension_cusp_forms(Gamma1(11),1)
0
sage: e = DirichletGroup(13).0
sage: e.order()
12
sage: dimension_cusp_forms(e,2)
0
sage: dimension_cusp_forms(e^2,2)
1

Check that github issue #12640 is fixed:

sage: dimension_cusp_forms(DirichletGroup(1)(1), 12)
1
sage: dimension_cusp_forms(DirichletGroup(2)(1), 24)
5

`sage.modular.dims.dimension_eis(X, k=2)`

The dimension of the space of Eisenstein series for the given congruence subgroup.

**INPUT:**

- `X` – congruence subgroup or Dirichlet character or integer
• $k$ – weight (integer)

EXAMPLES:

```
sage: from sage.modular.dims import dimension_eis
dimension_eis(5,4)
2
sage: dimension_eis(Gamma0(11),2)
1
sage: dimension_eis(Gamma1(13),2)
11
sage: dimension_eis(Gamma1(2006),2)
3711
sage: e = DirichletGroup(13).0
dimension_eis(e,2)
0
dimension_eis(e^2,2)
2
sage: e = DirichletGroup(13).0
dimension_eis(e,2)
0
dimension_eis(e^2,2)
2
dimension_eis(e,13)
2
sage: G = DirichletGroup(20)
dimension_eis(G.0,3)
4
dimension_eis(G.1,3)
6
dimension_eis(G.1^2,2)
6
sage: G = DirichletGroup(200)
e = prod(G.gens(), G(1))
e conductor()
200
dimension_eis(e,2)
4
sage: from sage.modular.dims import dimension_modular_forms
dimension_modular_forms(Gamma1(4), 11)
6
```

The dimension of the space of cusp forms for the given congruence subgroup (either $\Gamma_0(N)$, $\Gamma_1(N)$, or $\Gamma_H(N)$)
or Dirichlet character.

INPUT:

• \(X\) – congruence subgroup or Dirichlet character

• \(k\) – weight (integer)

EXAMPLES:

```
sage: from sage.modular.dims import dimension_modular_forms
sage: dimension_modular_forms(Gamma0(11),2)
2
sage: dimension_modular_forms(Gamma0(11),0)
1
sage: dimension_modular_forms(Gamma1(13),2)
13
sage: dimension_modular_forms(GammaH(11, [10]), 2)
10
sage: dimension_modular_forms(GammaH(11, [10]), 4)
20
sage: e = DirichletGroup(20).1
sage: dimension_modular_forms(e,3)
9
```

sage.modular.dims.dimension_new_cusp_forms\(X, k=2, p=0\)

Return the dimension of the new (or \(p\)-new) subspace of cusp forms for the character or group \(X\).

INPUT:

• \(X\) – integer, congruence subgroup or Dirichlet character

• \(k\) – weight (integer)

• \(p\) – 0 or a prime

EXAMPLES:

```
sage: from sage.modular.dims import dimension_new_cusp_forms
sage: dimension_new_cusp_forms(100,2)
1
sage: dimension_new_cusp_forms(Gamma0(100),2)
1
sage: dimension_new_cusp_forms(Gamma0(100),4)
5
sage: dimension_new_cusp_forms(Gamma1(100),2)
```

(continues on next page)
sage: dimension_new_cusp_forms(Gamma1(100), 4)
463
sage: dimension_new_cusp_forms(DirichletGroup(100).1^2, 2)
2
sage: dimension_new_cusp_forms(DirichletGroup(100).1^2, 4)
8
sage: sum(dimension_new_cusp_forms(e, 3) for e in DirichletGroup(30))
12
sage: dimension_new_cusp_forms(Gamma1(30), 3)
12

Check that github issue #12640 is fixed:

sage: dimension_new_cusp_forms(DirichletGroup(1)(1), 12)
1
sage: dimension_new_cusp_forms(DirichletGroup(2)(1), 24)
1

sage.modular.dims.eisen(p)

Return the Eisenstein number \( n \) which is the numerator of \( (p - 1)/12 \).

INPUT:

• \( p \) – a prime

OUTPUT: Integer

EXAMPLES:

sage: [(p, sage.modular.dims.eisen(p)) for p in prime_range(24)]
[(2, 1), (3, 1), (5, 1), (7, 1), (11, 5), (13, 1), (17, 4),
 (19, 3), (23, 11)]

sage.modular.dims.sturm_bound(level, weight=2)

Return the Sturm bound for modular forms with given level and weight.

For more details, see the documentation for the sturm_bound method of sage.modular.arithgroup.CongruenceSubgroup objects.

INPUT:

• \( \text{level} \) – an integer (interpreted as a level for Gamma0) or a congruence subgroup

• \( \text{weight} \) – an integer \( \geq 2 \) (default: 2)

EXAMPLES:

sage: from sage.modular.dims import sturm_bound
sage: sturm_bound(11,2)
2
sage: sturm_bound(389,2)
65
sage: sturm_bound(1,12)
4.4 Conjectural slopes of Hecke polynomials

Interface to Kevin Buzzard’s PARI program for computing conjectural slopes of characteristic polynomials of Hecke operators.

AUTHORS:

• William Stein (2006-03-05): Sage interface
• Kevin Buzzard: PARI program that implements underlying functionality

```
sage.modular.buzzard.buzzard_tpslopes(p, N, kmax)
```

Return a vector of length kmax, whose $k$'th entry ($0 \leq k \leq k_{\text{max}}$) is the conjectural sequence of valuations of eigenvalues of $T_p$ on forms of level $N$, weight $k$, and trivial character.

This conjecture is due to Kevin Buzzard, and is only made assuming that $p$ does not divide $N$ and if $p$ is $\Gamma_0(N)$-regular.

EXAMPLES:

```
sage: from sage.modular.buzzard import buzzard_tpslopes
sage: c = buzzard_tpslopes(2,1,50)
sage: c[50]
[4, 8, 13]
```

Hence Buzzard would conjecture that the 2-adic valuations of the eigenvalues of $T_2$ on cusp forms of level 1 and weight 50 are [4, 8, 13], which indeed they are, as one can verify by an explicit computation using, e.g., modular symbols:

```
sage: M = ModularSymbols(1,50, sign=1).cuspidal_submodule()
sage: T = M.hecke_operator(2)
sage: f = T.charpoly('x')
sage: f.newton_slopes(2)
[13, 8, 4]
```

AUTHORS:

• Kevin Buzzard: several PARI/GP scripts
• William Stein (2006-03-17): small Sage wrapper of Buzzard’s scripts

```
sage.modular.buzzard.gp()
```

Return a copy of the GP interpreter with the appropriate files loaded.

EXAMPLES:
4.5 Local components of modular forms

If \( f \) is a (new, cuspidal, normalised) modular eigenform, then one can associate to \( f \) an automorphic representation \( \pi_f \) of the group \( \text{GL}_2(\mathbb{A}) \) (where \( \mathbb{A} \) is the adele ring of \( \mathbb{Q} \)). This object factors as a restricted tensor product of components \( \pi_{f,v} \) for each place of \( \mathbb{Q} \). These are infinite-dimensional representations, but they are specified by a finite amount of data, and this module provides functions which determine a description of the local factor \( \pi_{f,p} \) at a finite prime \( p \).

The functions in this module are based on the algorithms described in [LW2012].

AUTHORS:

• David Loeffler
• Jared Weinstein

class sage.modular.local_comp.local_comp.ImprimitiveLocalComponent(newform, prime, twist_factor, min_twist, chi)

Bases: LocalComponentBase

A smooth representation which is not of minimal level among its character twists. Internally, this is stored as a pair consisting of a minimal local component and a character to twist by.

characters()

Return the pair of characters (either of \( \mathbb{Q}_p^* \) or of some quadratic extension) corresponding to this representation.

EXAMPLES:

```
sage: f = [f for f in Newforms(63, 4, names='a') if f[2] == 1][0]
sage: f.local_component(3).characters()
[Character of \mathbb{Q}_3^*, of level 1, mapping 2 |\rightarrow -1, 3 |\rightarrow d,
Character of \mathbb{Q}_3^*, of level 1, mapping 2 |\rightarrow -1, 3 |\rightarrow -d - 2]
```

check_tempered()

Check that this representation is quasi-tempered, i.e. \( \pi \otimes |\det |^{j/2} \) is tempered. It is well known that local components of modular forms are always tempered, so this serves as a useful check on our computations.

EXAMPLES:

```
sage: f = [f for f in Newforms(63, 4, names='a') if f[2] == 1][0]
sage: f.local_component(3).check_tempered()
```

is_primitive()

Return True if this local component is primitive (has minimal level among its character twists).

EXAMPLES:

```
sage: Newform("45a").local_component(3).is_primitive()
False
```
**minimal_twist()**

Return a twist of this local component which has the minimal possible conductor.

EXAMPLES:

```python
sage: Pi = Newform("75b").local_component(5)
sage: Pi.minimal_twist()
Smooth representation of GL_2(Q_5) with conductor 5^1
```

**species()**

The species of this local component, which is either ‘Principal Series’, ‘Special’ or ‘Supercuspidal’.

EXAMPLES:

```python
sage: Pi = Newform("45a").local_component(3)
sage: Pi.species()
'Special'
```

**twisting_character()**

Return the character giving the minimal twist of this representation.

EXAMPLES:

```python
sage: Pi = Newform("45a").local_component(3)
sage: Pi.twisting_character()
Dirichlet character modulo 3 of conductor 3 mapping 2 |--> -1
```

**sage.modular.local_comp.local_comp.LocalComponent(f, p, twist_factor=None)**

Calculate the local component at the prime \( p \) of the automorphic representation attached to the newform \( f \).

INPUT:

- \( f \) (Newform) a newform of weight \( k \geq 2 \)
- \( p \) (integer) a prime
- \( \text{twist_factor} \) (integer) an integer congruent to \( k \) modulo 2 (default: \( k-2 \))

**Note:** The argument \( \text{twist_factor} \) determines the choice of normalisation: if it is set to \( j \in \mathbb{Z} \), then the central character of \( \pi_{f, \ell} \) maps \( \ell \) to \( \ell^j \varepsilon(\ell) \) for almost all \( \ell \), where \( \varepsilon \) is the Nebentypus character of \( f \).

In the analytic theory it is conventional to take \( j = 0 \) (the “Langlands normalisation”), so the representation \( \pi_f \) is unitary; however, this is inconvenient for \( k \) odd, since in this case one needs to choose a square root of \( p \) and thus the map \( f \mapsto \pi_f \) is not Galois-equivariant. Hence we use, by default, the “Hecke normalisation” given by \( j = k-2 \). This is also the most natural normalisation from the perspective of modular symbols.

We also adopt a slightly unusual definition of the principal series: we define \( \pi(\chi_1, \chi_2) \) to be the induction from the Borel subgroup of the character of the maximal torus \((x, y) \mapsto \chi_1(x)\chi_2(y)|a|\), so its central character is \( z \mapsto \chi_1(z)\chi_2(z)|z| \). Thus \( \chi_1, \chi_2 \) is the restriction to \( \mathbb{Q}_p^\times \) of the unique character of the id'ele class group mapping \( \ell \) to \( \ell^{k-1}\varepsilon(\ell) \) for almost all \( \ell \). This has the property that the set \( \{\chi_1, \chi_2\} \) also depends Galois-equivariantly on \( f \).

EXAMPLES:
sage: Pi = LocalComponent(Newform('49a'), 7); Pi
Smooth representation of GL_2(Q_7) with conductor 7^2
sage: Pi.central_character()
Character of Q_7*, of level 0, mapping 7 |--> 1
sage: Pi.species()
'Supercuspidal'
sage: Pi.characters()
[Character of unramified extension Q_7(s)* (s^2 + 6*s + 3 = 0), of level 1, mapping...
  \rightarrow s |--> -d, 7 |--> 1,
Character of unramified extension Q_7(s)* (s^2 + 6*s + 3 = 0), of level 1, mapping...
  \rightarrow s |--> d, 7 |--> 1]

class sage.modular.local_comp.local_comp.LocalComponentBase(newform, prime, twist_factor)

Bases: SageObject

Base class for local components of newforms. Not to be directly instantiated; use the LocalComponent() constructor function.

central_character()

Return the central character of this representation. This is the restriction to \( Q_p^\times \) of the unique smooth character \( \omega \) of \( \mathbb{A}^\times / \mathbb{Q}^\times \) such that \( \omega(\varpi_\ell) = \ell^{\varepsilon(\ell)} \) for all primes \( \ell \mid N_p \), where \( \varpi_\ell \) is a uniformiser at \( \ell \), \( \varepsilon \) is the Nebentypus character of the newform \( f \), and \( j \) is the twist factor (see the documentation for LocalComponent()).

EXAMPLES:

sage: LocalComponent(Newform('27a'), 3).central_character()
Character of Q_3*, of level 0, mapping 3 |--> 1
sage: LocalComponent(Newforms(Gamma1(5), 5, names='c')[0], 5).central_character()
Character of Q_5*, of level 1, mapping 2 |--> c0 + 1, 5 |--> 125
sage: LocalComponent(Newforms(DirichletGroup(24)([1, -1, -1]), 3, names='a')[0], \omega \rightarrow 2).central_character()
Character of Q_2*, of level 3, mapping 7 |--> 1, 5 |--> -1, 2 |--> -2

check_tempered()

Check that this representation is quasi-tempered, i.e. \( \pi \otimes |det|^{j/2} \) is tempered. It is well known that local components of modular forms are always tempered, so this serves as a useful check on our computations.

EXAMPLES:

sage: from sage.modular.local_comp.local_comp import LocalComponentBase
sage: LocalComponentBase(Newform('50a'), 3, 0).check_tempered()
Traceback (most recent call last):
  ... NotImplementedError: <abstract method check_tempered at ...>

coefficient_field()

The field \( K \) over which this representation is defined. This is the field generated by the Hecke eigenvalues of the corresponding newform (over whatever base ring the newform is created).
EXAMPLES:

```python
sage: LocalComponent(Newforms(50)[0], 3).coefficient_field()
Rational Field
sage: LocalComponent(Newforms(Gamma1(10), 3, base_ring=QQbar)[0], 5).
˓→coefficient_field()
Algebraic Field
sage: LocalComponent(Newforms(DirichletGroup(5).0, 7,names='c')[0], 5).
˓→coefficient_field()
Number Field in c0 with defining polynomial x^2 + (5*zeta4 + 5)*x - 88*zeta4 
˓→over its base field
```

**conductor()**

The smallest $r$ such that this representation has a nonzero vector fixed by the subgroup \( \begin{pmatrix} * & * \\ 0 & 1 \end{pmatrix} \) (mod $p^r$).

This is equal to the power of $p$ dividing the level of the corresponding newform.

EXAMPLES:

```python
sage: LocalComponent(Newform('50a'), 5).conductor()
2
```

**newform()**

The newform of which this is a local component.

EXAMPLES:

```python
sage: LocalComponent(Newform('50a'), 5).newform()
q - q^2 + q^3 + q^4 + O(q^6)
```

**prime()**

The prime at which this is a local component.

EXAMPLES:

```python
sage: LocalComponent(Newform('50a'), 5).prime()
5
```

**species()**

The species of this local component, which is either ‘Principal Series’, ‘Special’ or ‘Supercuspidal’.

EXAMPLES:

```python
sage: from sage.modular.local_comp.local_comp import LocalComponentBase
sage: LocalComponentBase(Newform('50a'), 3, 0).species()
Traceback (most recent call last):
... Not Implemented Error: <abstract method species at ...>
```

**twist_factor()**

The unique $j$ such that \( \begin{pmatrix} p & 0 \\ 0 & p^j \end{pmatrix} \) acts as multiplication by $p^j$ times a root of unity.

There are various conventions for this; see the documentation of the `LocalComponent()` constructor function for more information.
The twist factor should have the same parity as the weight of the form, since otherwise the map sending $f$ to its local component won’t be Galois equivariant.

EXAMPLES:

```
sage: LocalComponent(Newforms(50)[0], 3).twist_factor()
0
sage: LocalComponent(Newforms(50)[0], 3, twist_factor=173).twist_factor()
173
```

class `sage.modular.local_comp.local_comp.PrimitiveLocalComponent`(*newform*, *prime*, *twist_factor*)

Bases: `LocalComponentBase`

Base class for primitive (twist-minimal) local components.

- **is_primitive()**
  - Return True if this local component is primitive (has minimal level among its character twists).

  EXAMPLES:

  ```sage```
  Newform("50a").local_component(5).is_primitive()
  True
  ```sage```

- **minimal_twist()**
  - Return a twist of this local component which has the minimal possible conductor.

  EXAMPLES:

  ```sage```
  Pi = Newform("50a").local_component(5)
  Pi.minimal_twist() == Pi
  True
  ```sage```

class `sage.modular.local_comp.local_comp.PrimitivePrincipalSeries`(*newform*, *prime*, *twist_factor*)

Bases: `PrincipalSeries`

A ramified principal series of the form $\pi(\chi_1, \chi_2)$ where $\chi_1$ is unramified but $\chi_2$ is not.

EXAMPLES:

```
sage: Pi = LocalComponent(Newforms(Gamma1(13), 2, names='a')[0], 13)
sage: type(Pi)
<class 'sage.modular.local_comp.local_comp.PrimitivePrincipalSeries'>
sage: TestSuite(Pi).run()
```

- **characters()**
  - Return the two characters $(\chi_1, \chi_2)$ such that the local component $\pi_{f,p}$ is the induction of the character $\chi_1 \times \chi_2$ of the Borel subgroup.

  EXAMPLES:

  ```sage```
  LocalComponent(Newforms(Gamma1(13), 2, names='a')[0], 13).characters()
  [Character of Q_13*, of level 0, mapping 13 |--> 3*a0 + 2,
  Character of Q_13*, of level 1, mapping 2 |--> a0 + 2, 13 |--> -3*a0 - 7]
  ```sage```
class sage.modular.local_comp.local_comp.PrimitiveSpecial(newform, prime, twist_factor)

Bases: PrimitiveLocalComponent

A primitive special representation: that is, the Steinberg representation twisted by an unramified character. All such representations have conductor 1.

EXAMPLES:

```
sage: Pi = LocalComponent(Newform('37a'), 37)
sage: Pi.species()
'Special'
sage: Pi.conductor()
1
sage: type(Pi)
<class 'sage.modular.local_comp.local_comp.PrimitiveSpecial'>
sage: TestSuite(Pi).run()
```

characters()

Return the defining characters of this representation. In this case, it will return the unique unramified character \( \chi \) of \( Q_p^\times \) such that this representation is equal to \( St \otimes \chi \), where \( St \) is the Steinberg representation (defined as the quotient of the parabolic induction of the trivial character by its trivial subrepresentation).

EXAMPLES:

Our first example is the newform corresponding to an elliptic curve of conductor 37. This is the nontrivial quadratic twist of Steinberg, corresponding to the fact that the elliptic curve has non-split multiplicative reduction at 37:

```
sage: LocalComponent(Newform('37a'), 37).characters()
[Character of Q_{37}^*, of level 0, mapping 37 |--> -1]
```

We try an example in odd weight, where the central character isn’t trivial:

```
sage: Pi = LocalComponent(Newforms(DirichletGroup(21)([-1, 1]), 3, names='j \rightarrow')[0], 7); Pi.characters()
[Character of Q_{7}^*, of level 0, mapping 7 |--> -1/2*j0^2 - 7/2]
sage: Pi.characters()[0]^2 == Pi.central_character()
True
```

An example using a non-standard twist factor:

```
sage: Pi = LocalComponent(Newforms(DirichletGroup(21)([-1, 1]), 3, names='j \rightarrow')[0], 7, twist_factor=3); Pi.characters()
[Character of Q_{7}^*, of level 0, mapping 7 |--> -7/2*j0^2 - 49/2]
sage: Pi.characters()[0]^2 == Pi.central_character()
True
```

check_tempered()

Check that this representation is tempered (after twisting by \( |det|^{j/2} \) where \( j \) is the twist factor). Since local components of modular forms are always tempered, this is a useful check on our calculations.

EXAMPLES:

```
sage: Pi = LocalComponent(Newforms(DirichletGroup(21)([-1, 1]), 3, names='j \rightarrow')[0], 7)
sage: Pi.check_tempered()
```
species()
The species of this local component, which is either ‘Principal Series’, ‘Special’ or ‘Supercuspidal’.

EXAMPLES:

```
sage: LocalComponent(Newform('37a'), 37).species()
'Special'
```

class sage.modular.local_comp.local_comp.PrimitiveSupercuspidal(newform, prime, twist_factor)

Bases: PrimitiveLocalComponent

A primitive supercuspidal representation.

Except for some exceptional cases when \( p = 2 \) which we do not implement here, such representations are parametrized by smooth characters of tamely ramified quadratic extensions of \( \mathbb{Q}_p \).

EXAMPLES:

```
sage: f = Newform("50a")

sage: Pi = LocalComponent(f, 5)

sage: type(Pi)
<class 'sage.modular.local_comp.local_comp.PrimitiveSupercuspidal'>

sage: Pi.species()
'Supercuspidal'

sage: TestSuite(Pi).run()
```

characters()
Return the two conjugate characters of \( K^\times \), where \( K \) is some quadratic extension of \( \mathbb{Q}_p \), defining this representation. An error will be raised in some 2-adic cases, since not all 2-adic supercuspidal representations arise in this way.

EXAMPLES:

The first example from [LW2012]:

```
sage: f = Newform('50a')

sage: Pi = LocalComponent(f, 5)

sage: chars = Pi.characters(); chars
[Character of unramified extension Q_5(s)* (s^2 + 4*s + 2 = 0), of level 1,\n  \rightarrow mapping s |--> -d - 1, 5 |--> 1,\nCharacter of unramified extension Q_5(s)* (s^2 + 4*s + 2 = 0), of level 1,\n  \rightarrow mapping s |--> d, 5 |--> 1]

sage: chars[0].base_ring()
Number Field in d with defining polynomial x^2 + x + 1
```

These characters are interchanged by the Frobenius automorphism of \( F_{25} \):

```
sage: chars[0] == chars[1]**5
True
```

A more complicated example (higher weight and nontrivial central character):

```
sage: f = Newforms(GammaH(25, [6]), 3, names='j')[0]; f
q + j0*q^2 + 1/3*j0*3*q^3 - 1/3*j0*2*q^4 + O(q^6)
```

(continues on next page)
sage: Pi = LocalComponent(f, 5)
sage: Pi.characters()
[Character of unramified extension $\mathbb{Q}_5(s)^\ast$ ($s^2 + 4s + 2 = 0$), of level 1,
  $\mapsto$ mapping $s \mapsto 1/3j0^2d - 1/3j0^3$, $5 \mapsto 5$,
  Character of unramified extension $\mathbb{Q}_5(s)^\ast$ ($s^2 + 4s + 2 = 0$), of level 1,
  $\mapsto$ mapping $s \mapsto -1/3j0^2d$, $5 \mapsto 5$]
sage: Pi.characters()[0].base_ring()
Number Field in d with defining polynomial $x^2 - j0x + 1/3j0^2$ over its base

Warning: The above output isn’t actually the same as in Example 2 of [LW2012], due to an error in the published paper (correction pending) – the published paper has the inverses of the above characters.

A higher level example:

sage: f = Newform('81a', names='j'); f
$q + j0q^2 + q^4 - j0q^5 + O(q^6)$
sage: LocalComponent(f, 3).characters()  # long time (12s on sage.math, 2012)
[Character of unramified extension $\mathbb{Q}_3(s)^\ast$ ($s^2 + 2s + 2 = 0$), of level 2,
  $\mapsto$ mapping $-2s \mapsto -2d + j0$, $4 \mapsto 1$, $3s + 1 \mapsto -j0d + 1$, $3 \mapsto 1$,
  Character of unramified extension $\mathbb{Q}_3(s)^\ast$ ($s^2 + 2s + 2 = 0$), of level 2,
  $\mapsto$ mapping $-2s \mapsto 2d - j0$, $4 \mapsto 1$, $3s + 1 \mapsto j0d - 2$, $3 \mapsto 1$]

Some ramified examples:

sage: Newform('27a').local_component(3).characters()
[Character of ramified extension $\mathbb{Q}_3(s)^\ast$ ($s^2 - 6 = 0$), of level 2, mapping $2 \mapsto 1$, $s + 1 \mapsto -d$, $s \mapsto -1$,
  Character of ramified extension $\mathbb{Q}_3(s)^\ast$ ($s^2 - 6 = 0$), of level 2, mapping $2 \mapsto 1$, $s + 1 \mapsto d - 1$, $s \mapsto -1$]
sage: LocalComponent(Newform('54a'), 3, twist_factor=4).characters()
[Character of ramified extension $\mathbb{Q}_3(s)^\ast$ ($s^2 - 3 = 0$), of level 2, mapping $2 \mapsto 1$, $s + 1 \mapsto -1/9d$, $s \mapsto -9$,
  Character of ramified extension $\mathbb{Q}_3(s)^\ast$ ($s^2 - 3 = 0$), of level 2, mapping $2 \mapsto 1$, $s + 1 \mapsto 1/9d - 1$, $s \mapsto -9$]

A 2-adic non-example:

sage: Newform('24a').local_component(2).characters()
Traceback (most recent call last):
... ValueError: Totally ramified 2-adic representations are not classified by... characters
Examples where $K^\times/Q^\times_p$ is not topologically cyclic (which complicates the computations greatly):

```python
sage: Newforms(DirichletGroup(64, QQ).1, 2, names='a')[0].local_component(2).characters() # long time, random
[Character of unramified extension Q_2(s)* (s^2 + s + 1 = 0), of level 3, mapping s |--> 1, 2*s + 1 |--> 1/2*a0, 4*s + 1 |--> -1, -1 |--> 1, 2 |--> 1, Character of unramified extension Q_2(s)* (s^2 + s + 1 = 0), of level 3, mapping s |--> 1, 2*s + 1 |--> 1/2*a0, 4*s + 1 |--> -1, -1 |--> 1, 2 |--> 1]

sage: Newform('243a', names='a').local_component(3).characters() # long time
[Character of ramified extension Q_3(s)* (s^2 - 6 = 0), of level 4, mapping -2*s |--> -d - 1, 4 |--> 1, 3*s + 1 |--> -d - 1, s |--> 1, Character of ramified extension Q_3(s)* (s^2 - 6 = 0), of level 4, mapping -2*s |--> d, 4 |--> 1, 3*s + 1 |--> d, s |--> 1]
```

`check_tempered()`

Check that this representation is tempered (after twisting by $|\det|^j/2$ where $j$ is the twist factor). Since local components of modular forms are always tempered, this is a useful check on our calculations.

Since the computation of the characters attached to this representation is not implemented in the odd-conductor case, a `NotImplementedError` will be raised for such representations.

EXAMPLES:

```python
sage: LocalComponent(Newform("50a"), 5).check_tempered()
sage: LocalComponent(Newform("27a"), 3).check_tempered()
```

`species()`

The species of this local component, which is either ‘Principal Series’, ‘Special’ or ‘Supercuspidal’.

EXAMPLES:

```python
sage: LocalComponent(Newform("49a"), 7).species()
'Supercuspidal'
```

`type_space()`

Return a `TypeSpace` object describing the (homological) type space of this newform, which we know is dual to the type space of the local component.

EXAMPLES:

```python
sage: LocalComponent(Newform("49a"), 7).type_space()
6-dimensional type space at prime 7 of form q + q^2 - q^4 + O(q^6)
```

**class** `sage.modular.local_comp.local_comp.PrincipalSeries(newform, prime, twist_factor)`

Bases: `PrimitiveLocalComponent`

A principal series representation. This is an abstract base class, not to be instantiated directly; see the subclasses `UnramifiedPrincipalSeries` and `PrimitivePrincipalSeries`.

`characters()`

Return the two characters $(\chi_1, \chi_2)$ such this representation $\pi_{f,p}$ is equal to the principal series $\pi(\chi_1, \chi_2)$.

EXAMPLES:
```python
sage: from sage.modular.local_comp.local_comp import PrincipalSeries
sage: PrincipalSeries(Newform('50a'), 3, 0).characters()
Traceback (most recent call last):
  ...  
NotImplementedError: <abstract method characters at ...>
```

```
check_tempered()

Check that this representation is tempered (after twisting by \(|\det|^{j/2}\), i.e. that \(|\chi_1(p)| = |\chi_2(p)| = p^{(j+1)/2}\). This follows from the Ramanujan–Petersson conjecture, as proved by Deligne.

EXAMPLES:
```
```
```python
sage: LocalComponent(Newform('49a'), 3).check_tempered()
```
```
species()

The species of this local component, which is either 'Principal Series', 'Special' or 'Supercuspidal'.

EXAMPLES:
```
```
```python
sage: LocalComponent(Newform('50a'), 3).species()
'Principal Series'
```
```
class sage.modular.local_comp.local_comp.UnramifiedPrincipalSeries(newform, prime, twist_factor)

Bases: PrincipalSeries

An unramified principal series representation of \(\text{GL}_2(\mathbb{Q}_p)\) (corresponding to a form whose level is not divisible by \(p\)).

EXAMPLES:
```
```
```python
sage: Pi = LocalComponent(Newform('50a'), 3)
sage: Pi.conductor()
0
sage: type(Pi)
<class 'sage.modular.local_comp.local_comp.UnramifiedPrincipalSeries'>
sage: TestSuite(Pi).run()
```
```
characters()

Return the two characters \((\chi_1, \chi_2)\) such this representation \(\pi_{f,p}\) is equal to the principal series \(\pi(\chi_1, \chi_2)\). These are the unramified characters mapping \(p\) to the roots of the Satake polynomial, so in most cases (but not always) they will be defined over an extension of the coefficient field of self.

EXAMPLES:
```
```
```python
sage: LocalComponent(Newform('11a'), 17).characters()
[  
Character of Q_{17}*, of level 0, mapping 17 |--> d,  
Character of Q_{17}*, of level 0, mapping 17 |--> -d - 2  
]
sage: LocalComponent(Newforms(Gamma1(5), 6, names='a')[1], 3).characters()
[  
Character of Q_{3}*, of level 0, mapping 3 |--> -3/2*a1 + 12,  
Character of Q_{3}*, of level 0, mapping 3 |--> -3/2*a1 - 12  
]
```
```
satake_polynomial()

Return the Satake polynomial of this representation, i.e. the polynomial whose roots are $\chi_1(p), \chi_2(p)$ where this representation is $\pi(\chi_1, \chi_2)$. Concretely, this is the polynomial

$$X^2 - p^{(j-k+2)/2} a_p(f) X + p^{j+1} \varepsilon(p).$$

An error will be raised if $j \neq k \mod 2$.

**EXAMPLES:**

```python
sage: LocalComponent(Newform('11a'), 17).satake_polynomial()
X^2 + 2*X + 17
sage: LocalComponent(Newform('11a'), 17, twist_factor = -2).satake_polynomial()
X^2 + 2/17*X + 1/17
```

### 4.6 Smooth characters of $p$-adic fields

Let $F$ be a finite extension of $\mathbb{Q}_p$. Then we may consider the group of smooth (i.e. locally constant) group homomorphisms $F^\times \to L^\times$, for $L$ any field. Such characters are important since they can be used to parametrise smooth representations of $GL_2(\mathbb{Q}_p)$, which arise as the local components of modular forms.

This module contains classes to represent such characters when $F$ is $\mathbb{Q}_p$ or a quadratic extension. In the latter case, we choose a quadratic extension $K$ of $\mathbb{Q}$ whose completion at $p$ is $F$, and use Sage’s wrappers of the Pari pari:idealstar and pari:ideallog methods to work in the finite group $\mathcal{O}_K/p^c$ for $c \geq 0$.

An example with characters of $\mathbb{Q}_7$:

```python
sage: from sage.modular.local_comp.smoothchar import SmoothCharacterGroupQp
sage: K.<z> = CyclotomicField(42)
.sage: G = SmoothCharacterGroupQp(7, K)
.sage: G.unit_gens(2), G.exponents(2)
([3, 7], [42, 0])
```

The output of the last line means that the group $\mathbb{Q}_7^\times/(1 + 7^2\mathbb{Z}_7)$ is isomorphic to $C_{42} \times \mathbb{Z}$, with the two factors being generated by 3 and 7 respectively. We create a character by specifying the images of these generators:

```python
sage: chi = G.character(2, [z^5, 11 + z]); chi
Character of $\mathbb{Q}_7^\times$, of level 2, mapping 3 |---> z^5, 7 |---> z + 11
sage: chi(4)
z^8
sage: chi(42)
z^10 + 11*z^9
```

Characters are themselves group elements, and basic arithmetic on them works:

```python
sage: chi**3
Character of $\mathbb{Q}_7^\times$, of level 2, mapping 3 |---> z^8 - z, 7 |---> z^3 + 33*z^2 + 363*z + 1331
sage: chi.multiplicative_order()
+Infinity
```

**class** `sage.modular.local_comp.smoothchar.SmoothCharacterGeneric(parent, c, values_on_gens)`

**Bases:** `MultiplicativeGroupElement`

A smooth (i.e. locally constant) character of $F^\times$, for $F$ some finite extension of $\mathbb{Q}_p$.  

Chapter 4. Miscellaneous Modules (to be sorted)
galois_conjugate()

Return the composite of this character with the order 2 automorphism of \( K/\mathbb{Q}_p \) (assuming \( K \) is quadratic).

Note that this is the Galois operation on the domain, not on the codomain.

EXAMPLES:

```python
sage: from sage.modular.local_comp.smoothchar import SmoothCharacterGroupUnramifiedQuadratic
sage: K.<w> = CyclotomicField(3)
sage: G = SmoothCharacterGroupUnramifiedQuadratic(2, K)
sage: chi = G.character(2, [w, -1, -1, 3*w])
sage: chi2 = chi.galois_conjugate(); chi2
Character of unramified extension \( \mathbb{Q}_2(s) \ast (s^2 + s + 1 = 0) \), of level 2, mapping \( s \mapsto -w - 1, 2s + 1 \mapsto 1, -1 \mapsto -1, 2 \mapsto 3w
```

```python
sage: chi.restrict_to_Qp() == chi2.restrict_to_Qp()
True
sage: chi * chi2 == chi.parent().compose_with_norm(chi.restrict_to_Qp())
True
```

level()

Return the level of this character, i.e. the smallest integer \( c \geq 0 \) such that it is trivial on \( 1 + p^c \).

EXAMPLES:

```python
sage: from sage.modular.local_comp.smoothchar import SmoothCharacterGroupQp
sage: SmoothCharacterGroupQp(7, QQ).character(2, [-1, 1]).level()
1
```

multiplicative_order()

Return the order of this character as an element of the character group.

EXAMPLES:

```python
sage: from sage.modular.local_comp.smoothchar import SmoothCharacterGroupQp
sage: K.<z> = CyclotomicField(42)
sage: G = SmoothCharacterGroupQp(7, K)
sage: G.character(3, [z^10 - z^3, 11]).multiplicative_order()
+Infinity
sage: G.character(3, [z^10 - z^3, 1]).multiplicative_order()
42
sage: G.character(1, [z^7, z^14]).multiplicative_order()
6
sage: G.character(0, [1]).multiplicative_order()
1
```

restrict_to_Qp()

Return the restriction of this character to \( \mathbb{Q}_p^\times \), embedded as a subfield of \( F^\times \).

EXAMPLES:

```python
sage: from sage.modular.local_comp.smoothchar import SmoothCharacterGroupRamifiedQuadratic
sage: SmoothCharacterGroupRamifiedQuadratic(3, 0, QQ).character(0, [2]).
```

(continues on next page)
class sage.modular.local_comp.smoothchar.SmoothCharacterGroupGeneric(p, base_ring)

Bases: Parent

The group of smooth (i.e. locally constant) characters of a $p$-adic field, with values in some ring $R$. This is an abstract base class and should not be instantiated directly.

Element

alias of SmoothCharacterGeneric

base_extend(ring)

Return the character group of the same field, but with values in a new coefficient ring into which the old coefficient ring coerces. An error will be raised if there is no coercion map from the old coefficient ring to the new one.

EXAMPLES:

```sage
from sage.modular.local_comp.smoothchar import SmoothCharacterGroupQp
G = SmoothCharacterGroupQp(3, QQ)
G.base_extend(QQbar)
Group of smooth characters of $\mathbb{Q}_3^*$ with values in Algebraic Field
G.base_extend(Zmod(3))
Traceback (most recent call last):
... TypeError: no canonical coercion from Rational Field to Ring of integers modulo 3
```

change_ring(ring)

Return the character group of the same field, but with values in a different coefficient ring. To be implemented by all derived classes (since the generic base class can’t know the parameters).

EXAMPLES:

```sage
from sage.modular.local_comp.smoothchar import SmoothCharacterGroupGeneric
SmoothCharacterGroupGeneric(3, QQ).change_ring(ZZ)
Traceback (most recent call last):
... NotImplementedError: <abstract method change_ring at ...
```

class sage.modular.local_comp.smoothchar.SmoothCharacterGroupQp

The group of smooth (i.e. locally constant) characters of a $p$-adic field, with values in some ring $R$. This is an abstract base class and should not be instantiated directly.

Element

alias of SmoothCharacterGeneric

restrict_to_Qp()

Character of $\mathbb{Q}_3^*$, of level 0, mapping 3 $\rightarrow$ 4

character(level, values_on_gens)

Return the unique character of the given level whose values on the generators returned by self. unit_gens(level) are values_on_gens.

INPUT:

- level (integer) an integer $\geq 0$
- values_on_gens (sequence) a sequence of elements of length equal to the length of self. unit_gens(level). The values should be convertible (that is, possibly noncanonically) into the base ring of self; they should all be units, and all but the last must be roots of unity (of the orders given by self.exponents(level)).

Note: The character returned may have level less than level in general.
EXAMPLES:

```
sage: from sage.modular.local_comp.smoothchar import SmoothCharacterGroupQp
sage: K.<z> = CyclotomicField(42)
sage: G = SmoothCharacterGroupQp(7, K)
sage: G.character(2, [z^6, 8])
Character of Q_7*, of level 2, mapping 3 |--> z^6, 7 |--> 8
sage: G.character(2, [z^7, 8])
Character of Q_7*, of level 1, mapping 3 |--> z^7, 7 |--> 8
```

Non-examples:

```
sage: G.character(1, [z, 1])
Traceback (most recent call last):
  ... ValueError: value on generator 3 (=z) should be a root of unity of order 6
sage: G.character(1, [1, 0])
Traceback (most recent call last):
  ... ValueError: value on uniformiser 7 (=0) should be a unit
```

An example with a funky coefficient ring:

```
sage: G = SmoothCharacterGroupQp(7, Zmod(9))
sage: G.character(1, [2, 2])
Character of Q_7* of level 1, mapping 3 |--> 2, 7 |--> 2
sage: G.character(1, [2, 3])
Traceback (most recent call last):
  ... ValueError: value on uniformiser 7 (=3) should be a unit
```

```
compose_with_norm(chi)
Calculate the character of $K^\times$ given by $\chi \circ \text{Norm}_{K/Q_p}$. Here $K$ should be a quadratic extension and $\chi$ a character of $Q_p^\times$.
EXAMPLES:
When $K$ is the unramified quadratic extension, the level of the new character is the same as the old:
```
sage: from sage.modular.local_comp.smoothchar import SmoothCharacterGroupQp,
   SmoothCharacterGroupRamifiedQuadratic, SmoothCharacterGroupUnramifiedQuadratic
sage: K.<w> = CyclotomicField(6)
sage: G = SmoothCharacterGroupQp(3, K)
sage: chi = G.character(2, [w, 5])
sage: H = SmoothCharacterGroupUnramifiedQuadratic(3, K)
sage: H.compose_with_norm(chi)
Character of unramified extension Q_3(s)* (s^2 + 2*s + 2 = 0), of level 2, mapping -2*s |--> -1, 4 |--> -w, 3*s + 1 |--> w - 1, 3 |--> 25
```

In ramified cases, the level of the new character may be larger:
```
sage: H = SmoothCharacterGroupRamifiedQuadratic(3, 0, K)
sage: H.compose_with_norm(chi)
Character of ramified extension Q_3(s)* (s^2 - 3 = 0), of level 3, mapping 2 |---> w - 1, s + 1 |--> -w, s |--> -5
```

4.6. Smooth characters of $p$-adic fields
On the other hand, since norm is not surjective, the result can even be trivial:

\[
\text{sage: chi = G.character(1, [-1, -1]); chi}
\]
Character of Q_3*, of level 1, mapping 2 |--> -1, 3 |--> -1

\[
\text{sage: H.compose_with_norm(chi)}
\]
Character of ramified extension Q_3(s)* (s^2 - 3 = 0), of level 0, mapping s |---> 1

**discrete_log**(level)
Given an element \(x \in F^\times\) (lying in the number field \(K\) of which \(F\) is a completion, see module docstring), express the class of \(x\) in terms of the generators of \(F^\times/(1 + p^e)^\times\) returned by `unit_gens()`.

This should be overridden by all derived classes. The method should first attempt to canonically coerce \(x\) into `self.number_field()`, and check that the result is not zero.

**EXAMPLES:**

\[
\text{sage: from sage.modular.local_comp.smoothchar import SmoothCharacterGroupGeneric}
\]
\[
\text{sage: SmoothCharacterGroupGeneric(3, QQ).discrete_log(3)}
\]
Traceback (most recent call last):
... NotImplementedError: <abstract method discrete_log at ...>

**exponents**(level)
The orders \(n_1, \ldots, n_d\) of the generators \(x_i\) of \(F^\times/(1 + p^e)^\times\) returned by `unit_gens()`.

**EXAMPLES:**

\[
\text{sage: from sage.modular.local_comp.smoothchar import SmoothCharacterGroupGeneric}
\]
\[
\text{sage: SmoothCharacterGroupGeneric(3, QQ).exponents(3)}
\]
Traceback (most recent call last):
... NotImplementedError: <abstract method exponents at ...>

**ideal**(level)
Return the \(level\)-th power of the maximal ideal of the ring of integers of the \(p\)-adic field. Since we approximate by using number field arithmetic, what is actually returned is an ideal in a number field.

**EXAMPLES:**

\[
\text{sage: from sage.modular.local_comp.smoothchar import SmoothCharacterGroupGeneric}
\]
\[
\text{sage: SmoothCharacterGroupGeneric(3, QQ).ideal(3)}
\]
Traceback (most recent call last):
... NotImplementedError: <abstract method ideal at ...>

**norm_character()**
Return the normalised absolute value character in this group (mapping a uniformiser to \(1/q\) where \(q\) is the order of the residue field).

**EXAMPLES:**

\[
\text{sage: from sage.modular.local_comp.smoothchar import SmoothCharacterGroupQp, UnramifiedQuadratic}
\]
\[
\text{sage: SmoothCharacterGroupQp(5, QQ).norm_character()}
\]
Character of Q_5*, of level 0, mapping 5 |---> 1/5

(continues on next page)
sage: SmoothCharacterGroupUnramifiedQuadratic(2, QQ).norm_character()
Character of unramified extension $\mathbb{Q}_2(s)^*$ ($s^2 + s + 1 = 0$), of level 0,
→ mapping 2 |--> 1/4

prime()

The residue characteristic of the underlying field.

EXAMPLES:

sage: from sage.modular.local_comp.smoothchar import SmoothCharacterGroupGeneric
sage: SmoothCharacterGroupGeneric(3, QQ).prime()
3

subgroup_gens(level)

A set of elements of $(\mathcal{O}_F/p^c)^\times$ generating the kernel of the reduction map to $(\mathcal{O}_F/p^{c-1})^\times$.

EXAMPLES:

sage: from sage.modular.local_comp.smoothchar import SmoothCharacterGroupGeneric
sage: SmoothCharacterGroupGeneric(3, QQ).subgroup_gens(3)
Traceback (most recent call last):
  ... Not Implemented Error: <abstract method subgroup_gens at ...>

unit_gens(level)

A list of generators $x_1, \ldots, x_d$ of the abelian group $F^\times/(1 + p^c)^\times$, where $c$ is the given level, satisfying no relations other than $x_i^{n_i} = 1$ for each $i$ (where the integers $n_i$ are returned by exponents()). We adopt the convention that the final generator $x_d$ is a uniformiser (and $n_d = 0$).

EXAMPLES:

sage: from sage.modular.local_comp.smoothchar import SmoothCharacterGroupGeneric
sage: SmoothCharacterGroupGeneric(3, QQ).unit_gens(3)
Traceback (most recent call last):
  ... Not Implemented Error: <abstract method unit_gens at ...>

class sage.modular.local_comp.smoothchar.SmoothCharacterGroupQp(p, base_ring)

Bases: SmoothCharacterGroupGeneric

The group of smooth characters of $\mathbb{Q}_p^\times$, with values in some fixed base ring.

EXAMPLES:

sage: from sage.modular.local_comp.smoothchar import SmoothCharacterGroupQp
sage: G = SmoothCharacterGroupQp(7, QQ); G
Group of smooth characters of $\mathbb{Q}_7^\times$ with values in Rational Field
sage: TestSuite(G).run()
sage: G == loads(dumps(G))
True

change_ring(ring)

Return the group of characters of the same field but with values in a different ring. This need not have anything to do with the original base ring, and in particular there won’t generally be a coercion map from self to the new group – use base_extend() if you want this.

4.6. Smooth characters of $p$-adic fields
EXAMPLES:

```
sage: from sage.modular.local_comp.smoothchar import SmoothCharacterGroupQp
sage: SmoothCharacterGroupQp(7, Zmod(3)).change_ring(CC)
Group of smooth characters of Q_7* with values in Complex Field with 53 bits of precision
```

discrete_log(level, x)
Express the class of \( x \) in \( Q_p^\times/(1+p^e)^\times \) in terms of the generators returned by \( \text{unit}_\text{gens}() \).

EXAMPLES:

```
sage: from sage.modular.local_comp.smoothchar import SmoothCharacterGroupQp
sage: G = SmoothCharacterGroupQp(7, QQ)
sage: G.discrete_log(0, 14)
[1]
sage: G.discrete_log(1, 14)
[2, 1]
sage: G.discrete_log(5, 14)
[9308, 1]
```

exponents(level)
Return the exponents of the generators returned by \( \text{unit}_\text{gens}() \).

EXAMPLES:

```
sage: from sage.modular.local_comp.smoothchar import SmoothCharacterGroupQp
sage: SmoothCharacterGroupQp(7, QQ).exponents(3)
[294, 0]
sage: SmoothCharacterGroupQp(2, QQ).exponents(4)
[2, 4, 0]
```

from_dirichlet(chi)
Given a Dirichlet character \( \chi \), return the factor at \( p \) of the adelic character \( \phi \) which satisfies \( \phi(\wp_\ell) = \chi(\ell) \) for almost all \( \ell \), where \( \wp_\ell \) is a uniformizer at \( \ell \).

More concretely, if we write \( \chi = \chi_p \chi_M \) as a product of characters of \( p \)-power, resp prime-to-\( p \), conductor, then this function returns the character of \( Q_p^\times \) sending \( p \) to \( \chi_M(p) \) and agreeing with \( \chi_p^{-1} \) on integers that are 1 modulo \( M \) and coprime to \( p \).

EXAMPLES:

```
sage: from sage.modular.local_comp.smoothchar import SmoothCharacterGroupQp
sage: G = SmoothCharacterGroupQp(3, CyclotomicField(6))
sage: G.from_dirichlet(DirichletGroup(9).0)
Character of Q_3*, of level 2, mapping 2 |--> -zeta6 + 1, 3 |--> 1
```

ideal(level)
Return the \( 1 \)-\( level \)-th power of the maximal ideal. Since we approximate by using rational arithmetic, what is actually returned is an ideal of \( \mathbb{Z} \).

EXAMPLES:

```
sage: from sage.modular.local_comp.smoothchar import SmoothCharacterGroupQp
sage: SmoothCharacterGroupQp(7, Zmod(3)).ideal(2)
Principal ideal (49) of Integer Ring
```
**number_field()**
Return the number field used for calculations (a dense subfield of the local field of which this is the character group). In this case, this is always the rational field.

**EXAMPLES:**
```python
sage: from sage.modular.local_comp.smoothchar import SmoothCharacterGroupQp
sage: SmoothCharacterGroupQp(7, Zmod(3)).number_field()
Rational Field
```

**quadratic_chars()**
Return a list of the (non-trivial) quadratic characters in this group. This will be a list of 3 characters, unless \( p = 2 \) when there are 7.

**EXAMPLES:**
```python
sage: from sage.modular.local_comp.smoothchar import SmoothCharacterGroupQp
sage: SmoothCharacterGroupQp(7, QQ).quadratic_chars()
[Character of Q_7*, of level 0, mapping 7 |--> -1,
 Character of Q_7*, of level 1, mapping 3 |--> -1, 7 |--> -1,
 Character of Q_7*, of level 1, mapping 3 |--> -1, 7 |--> 1]
sage: SmoothCharacterGroupQp(2, QQ).quadratic_chars()
[Character of Q_2*, of level 0, mapping 2 |--> -1,
 Character of Q_2*, of level 2, mapping 3 |--> -1, 2 |--> -1,
 Character of Q_2*, of level 2, mapping 3 |--> -1, 2 |--> 1,
 Character of Q_2*, of level 3, mapping 7 |--> -1, 5 |--> -1, 2 |--> -1,
 Character of Q_2*, of level 3, mapping 7 |--> -1, 5 |--> -1, 2 |--> 1,
 Character of Q_2*, of level 3, mapping 7 |--> 1, 5 |--> -1, 2 |--> -1,
 Character of Q_2*, of level 3, mapping 7 |--> 1, 5 |--> -1, 2 |--> 1]
```

**subgroup_gens**(level)
Return a list of generators for the kernel of the map \((\mathbb{Z}_p/p^c)^\times \rightarrow (\mathbb{Z}_p/p^{c-1})^\times\).

**INPUT:**
- \( c \) (integer) an integer \( \geq 1 \)

**EXAMPLES:**
```python
sage: from sage.modular.local_comp.smoothchar import SmoothCharacterGroupQp
sage: G = SmoothCharacterGroupQp(7, QQ)
sage: G.subgroup_gens(1)
[3]
sage: G.subgroup_gens(2)
[8]
sage: G = SmoothCharacterGroupQp(2, QQ)
sage: G.subgroup_gens(1)
[]
sage: G.subgroup_gens(2)
[3]
sage: G.subgroup_gens(3)
[5]
```

**unit_gens**(level)
Return a set of generators \( x_1, \ldots, x_d \) for \( \mathbb{Q}_p^\times / (1 + p^c \mathbb{Z}_p)^\times \). These must be independent in the sense that
there are no relations between them other than relations of the form $x_i^{n_i} = 1$. They need not, however, be in Smith normal form.

EXAMPLES:

```python
sage: from sage.modular.local_comp.smoothchar import SmoothCharacterGroupQp
sage: SmoothCharacterGroupQp(7, QQ).unit gens(3)
[3, 7]
sage: SmoothCharacterGroupQp(2, QQ).unit gens(4)
[15, 5, 2]
```

class sage.modular.local_comp.smoothchar.SmoothCharacterGroupQuadratic($p$, base_ring)

Bases: SmoothCharacterGroupGeneric

The group of smooth characters of $E^\times$, where $E$ is a quadratic extension of $\mathbb{Q}_p$.

```python
discrete_log($level$, $x$, gens=None)
Express the class of $x$ in $F^\times/(1+p^e)^\times$ in terms of the generators returned by self.unit gens($level$), or a custom set of generators if given.

EXAMPLES:

```python
sage: from sage.modular.local_comp.smoothchar import...
...
from sage.modular.local_comp.smoothchar import
sage: G = SmoothCharacterGroupUnramifiedQuadratic(2, QQ)
sage: G.discrete_log(0, 12)
[2]
sage: G.discrete_log(1, 12)
[0, 2]
sage: v = G.discrete_log(5, 12); v
[0, 2, 0, 1, 2]
sage: g = G.unit gens(5); prod([g[i]**v[i] for i in [0..4]])/12 - 1 in G.
   ideal(5)
True
sage: G.discrete_log(3, G.number field()([1,1]))
[2, 0, 0, 1, 0]
sage: H = SmoothCharacterGroupUnramifiedQuadratic(5, QQ)
sage: x = H.number field()([1,1]); x
s + 1
sage: v = H.discrete_log(5, x); v
[22, 263, 379, 0]
sage: h = H.unit gens(5); prod([h[i]**v[i] for i in [0..3]])/x - 1 in H.ideal(5)
True
```

An example with a custom generating set:
extend_character(level, chi, vals, check=True)

Return the unique character of $F^\times$ which coincides with $\chi$ on $Q_p^\times$ and maps the generators of the quotient returned by quotient_gens() to vals.

INPUT:

- chi: a smooth character of $Q_p$, where $p$ is the residue characteristic of $F$, with values in the base ring of self (or some other ring coercible to it)
- level: the level of the new character (which should be at least the level of chi)
- vals: a list of elements of the base ring of self (or some other ring coercible to it), specifying values on the quotients returned by quotient_gens().

A ValueError will be raised if $x^t \neq \chi(\alpha^t)$, where $t$ is the smallest integer such that $\alpha^t$ is congruent modulo $p^{\text{level}}$ to an element of $Q_p$.

EXAMPLES:

We extend an unramified character of $Q_5^\times$ to the unramified quadratic extension in various ways.

```python
sage: from sage.modular.local_comp.smoothchar import SmoothCharacterGroupQp,
    SmoothCharacterGroupUnramifiedQuadratic
sage: chi = SmoothCharacterGroupQp(5, QQ).character(0, [7]); chi
Character of $Q_5^\times$, of level 0, mapping 5 |--> 7
sage: G = SmoothCharacterGroupUnramifiedQuadratic(5, QQ)
sage: G.extend_character(1, chi, [-1])
Character of unramified extension $Q_5(s)^\times$ ($s^2 + 4s + 2 = 0$), of level 1, mapping $s$ |--> -1, 5 |--> 7
sage: G.extend_character(2, chi, [-1])
Character of unramified extension $Q_5(s)^\times$ ($s^2 + 4s + 2 = 0$), of level 1, mapping $s$ |--> -1, 5 |--> 7
sage: G.extend_character(3, chi, [1])
Character of unramified extension $Q_5(s)^\times$ ($s^2 + 4s + 2 = 0$), of level 0, mapping 5 |--> 7
sage: K.<z> = CyclotomicField(6); G.base_extend(K).extend_character(1, chi, [z])
Character of unramified extension $Q_5(s)^\times$ ($s^2 + 4s + 2 = 0$), of level 1, mapping $s$ |--> -z + 1, 5 |--> 7
```

Extensions of higher level:

```python
sage: K.<z> = CyclotomicField(20); rho = G.base_extend(K).extend_character(2, chi, [z]); rho
Character of unramified extension $Q_5(s)^\times$ ($s^2 + 4s + 2 = 0$), of level 2, mapping $11s - 10$ |--> $z^5$, 6 |--> 1, $5s + 1$ |--> $z^4$, 5 |--> 7
```

(continues on next page)
Examples where it doesn’t work:

```
sage: G = SmoothCharacterGroupQp(2, QQ); H = SmoothCharacterGroupUnramifiedQuadratic(2, QQ)
sage: chi = G.character(3, [1, -1, 7])
sage: H.extend_character(2, chi, [-1])
Traceback (most recent call last):
  ... ValueError: Level of extended character cannot be smaller than level of character of Qp
```

**quotient_gens**(n)

Return a list of elements of $E$ which are a generating set for the quotient $E^\times / Q_p^\times$, consisting of elements which are “minimal” in the sense of [LW12].

In the examples we implement here, this quotient is almost always cyclic: the exceptions are the unramified quadratic extension of $Q_2$ for $n \geq 3$, and the extension $Q_3(\sqrt{-3})$ for $n \geq 4$.

**EXAMPLES:**

```
sage: from sage.modular.local_comp.smoothchar import SmoothCharacterGroupUnramifiedQuadratic
sage: G = SmoothCharacterGroupUnramifiedQuadratic(7, QQ)
sage: G.quotient_gens(1)
[2*s - 2]
sage: G.quotient_gens(2)
[15*s + 21]
sage: G.quotient_gens(3)
[-75*s + 33]
```

A ramified case:

```
sage: from sage.modular.local_comp.smoothchar import SmoothCharacterGroupRamifiedQuadratic
sage: G = SmoothCharacterGroupRamifiedQuadratic(7, 0, QQ)
sage: G.quotient_gens(3)
[22*s + 21]
```

An example where the quotient group is not cyclic:

```
sage: G = SmoothCharacterGroupUnramifiedQuadratic(2, QQ)
sage: G.quotient_gens(1)
[s + 1]
sage: G.quotient_gens(2)
[-s + 2]
```
sage: G.quotientgens(3)
[-17*s - 14, 3*s - 2]

class sage.modular.local_comp.smoothchar.SmoothCharacterGroupRamifiedQuadratic(prime, flag, base_ring, names='s')

Bases: SmoothCharacterGroupQuadratic

The group of smooth characters of $K^\times$, where $K$ is a ramified quadratic extension of $\mathbb{Q}_p$, and $p \neq 2$.

change_ring(ring)

Return the character group of the same field, but with values in a different coefficient ring. This need not have anything to do with the original base ring, and in particular there won’t generally be a coercion map from self to the new group – use base_extend() if you want this.

EXAMPLES:

sage: from sage.modular.local_comp.smoothchar import SmoothCharacterGroupRamifiedQuadratic
sage: G = SmoothCharacterGroupRamifiedQuadratic(5, 1, QQ, 'a'); G.change_ring(CC)
Group of smooth characters of ramified extension $\mathbb{Q}_5(a)^*$ with values in Complex Field with 53 bits of precision

exponents(c)

Return the orders of the independent generators of the unit group returned by unit_gens().

EXAMPLES:

sage: from sage.modular.local_comp.smoothchar import SmoothCharacterGroupRamifiedQuadratic
sage: G = SmoothCharacterGroupRamifiedQuadratic(5, 0, QQ)
sage: G.exponents(0)
(0,)
sage: G.exponents(1)
(4, 0)
sage: G.exponents(8)
(500, 625, 0)

ideal(c)

Return the ideal $p^c$ of self.number_field(). The result is cached, since we use the methods idealstar() and ideallog() which cache a Pari bid structure.

EXAMPLES:

sage: from sage.modular.local_comp.smoothchar import SmoothCharacterGroupRamifiedQuadratic
sage: G = SmoothCharacterGroupRamifiedQuadratic(5, 1, QQ, 'a'); I = G.ideal(3);
Fractional ideal (25, 5*a)
sage: I is G.ideal(3)
True
**number_field()**

Return a number field of which this is the completion at $p$.

**EXAMPLES:**

```python
sage: from sage.modular.local_comp.smoothchar import SmoothCharacterGroupRamifiedQuadratic
    sage: SmoothCharacterGroupRamifiedQuadratic(7, 0, QQ, 'a').number_field()
Number Field in a with defining polynomial x^2 - 7
sage: SmoothCharacterGroupRamifiedQuadratic(5, 1, QQ, 'b').number_field()
Number Field in b with defining polynomial x^2 - 10
sage: SmoothCharacterGroupRamifiedQuadratic(7, 1, Zmod(6), 'c').number_field()
Number Field in c with defining polynomial x^2 - 35
```

**subgroup_gens(level)**

A set of elements of $(\mathcal{O}_F/p^c)\times$ generating the kernel of the reduction map to $(\mathcal{O}_F/p^{c-1})\times$.

**EXAMPLES:**

```python
sage: from sage.modular.local_comp.smoothchar import SmoothCharacterGroupRamifiedQuadratic
    sage: G = SmoothCharacterGroupRamifiedQuadratic(3, 1, QQ)
sage: G.subgroup_gens(2)
[s + 1]
```

**unit_gens(c)**

A list of generators $x_1, \ldots, x_d$ of the abelian group $F^\times/(1 + p^c)^\times$, where $c$ is the given level, satisfying no relations other than $x_i^{n_i} = 1$ for each $i$ (where the integers $n_i$ are returned by `exponents()`). We adopt the convention that the final generator $x_d$ is a uniformiser.

**EXAMPLES:**

```python
sage: from sage.modular.local_comp.smoothchar import SmoothCharacterGroupRamifiedQuadratic
    sage: G = SmoothCharacterGroupRamifiedQuadratic(5, 0, QQ)
sage: G.unit_gens(0)
[s]
sage: G.unit_gens(1)
[2, s]
sage: G.unit_gens(8)
[2, s + 1, s]
```

**class sage.modular.local_comp.smoothchar.SmoothCharacterGroupUnramifiedQuadratic**

Bases: `SmoothCharacterGroupQuadratic`

The group of smooth characters of $\mathbb{Q}_p^\times$, where $\mathbb{Q}_p^\times$ is the unique unramified quadratic extension of $\mathbb{Q}_p$. We represent $\mathbb{Q}_p^\times$ internally as the completion at the prime above $p$ of a quadratic number field, defined by (the obvious lift to $\mathbb{Z}$ of) the Conway polynomial modulo $p$ of degree 2.

**EXAMPLES:**

```python
sage: from sage.modular.local_comp.smoothchar import SmoothCharacterGroupUnramifiedQuadratic
    sage: SmoothCharacterGroupUnramifiedQuadratic(3, 0, QQ)
```
sage: G = SmoothCharacterGroupUnramifiedQuadratic(3, QQ); G
Group of smooth characters of unramified extension $\mathbb{Q}_3(s^*)$ ($s^2 + 2s + 2 = 0$) with $s$-values in Rational Field
sage: G.unit_gens(3)
[-11*s, 4, 3*s + 1, 3]
sage: TestSuite(G).run()
sage: TestSuite(SmoothCharacterGroupUnramifiedQuadratic(2, QQ)).run()

change_ring(ring)

Return the character group of the same field, but with values in a different coefficient ring. This need not have anything to do with the original base ring, and in particular there won't generally be a coercion map from self to the new group -- use base_extend() if you want this.

EXAMPLES:

sage: from sage.modular.local_comp.smoothchar import SmoothCharacterGroupUnramifiedQuadratic
sage: SmoothCharacterGroupUnramifiedQuadratic(7, Zmod(3), names='foo').change_ring(CC)
Group of smooth characters of unramified extension $\mathbb{Q}_7(foo^*)$ ($foo^2 + 6foo + 3 = 0$) with values in Complex Field with 53 bits of precision

exponents(c)

The orders $n_1, \ldots, n_d$ of the generators $x_i$ of $F^\times/(1+p^c)^\times$ returned by unit_gens().

EXAMPLES:

sage: from sage.modular.local_comp.smoothchar import SmoothCharacterGroupUnramifiedQuadratic
sage: SmoothCharacterGroupUnramifiedQuadratic(7, QQ).exponents(2)
[48, 7, 7, 0]
sage: SmoothCharacterGroupUnramifiedQuadratic(2, QQ).exponents(3)
[3, 4, 2, 2, 0]
sage: SmoothCharacterGroupUnramifiedQuadratic(2, QQ).exponents(2)
[3, 2, 2, 0]

ideal(c)

Return the ideal $p^c$ of self.number_field(). The result is cached, since we use the methods idealstar() and ideallog() which cache a Pari bid structure.

EXAMPLES:

sage: from sage.modular.local_comp.smoothchar import SmoothCharacterGroupUnramifiedQuadratic
sage: G = SmoothCharacterGroupUnramifiedQuadratic(7, QQ, 'a'); I = G.ideal(3); I
Fractional ideal (343)
sage: I is G.ideal(3)
True

number_field()

Return a number field of which this is the completion at $p$, defined by a polynomial whose discriminant is not divisible by $p$.

EXAMPLES:
sage: from sage.modular.local_comp.smoothchar import SmoothCharacterGroupUnramifiedQuadratic
sage: SmoothCharacterGroupUnramifiedQuadratic(7, QQ, 'a').number_field()
Number Field in a with defining polynomial x^2 + 6*x + 3
sage: SmoothCharacterGroupUnramifiedQuadratic(5, QQ, 'b').number_field()
Number Field in b with defining polynomial x^2 + 4*x + 2
sage: SmoothCharacterGroupUnramifiedQuadratic(2, QQ, 'c').number_field()
Number Field in c with defining polynomial x^2 + x + 1

subgroup_gens(level)
A set of elements of \((\mathcal{O}_F/p^c)^\times\) generating the kernel of the reduction map to \((\mathcal{O}_F/p^{c-1})^\times\).

EXAMPLES:

sage: from sage.modular.local_comp.smoothchar import SmoothCharacterGroupUnramifiedQuadratic
sage: SmoothCharacterGroupUnramifiedQuadratic(7, QQ).subgroup_gens(1)
[s]
sage: SmoothCharacterGroupUnramifiedQuadratic(7, QQ).subgroup_gens(2)
[8, 7*s + 1]
sage: SmoothCharacterGroupUnramifiedQuadratic(7, QQ).subgroup_gens(3)
[22*s, 8, 7*s + 1, 7]
sage: SmoothCharacterGroupUnramifiedQuadratic(7, QQ).subgroup_gens(4)
[169*s + 49, 8, 7*s + 1, 7]

In the 2-adic case there can be more than 4 generators:

sage: SmoothCharacterGroupUnramifiedQuadratic(2, QQ).subgroup_gens(0)
[2]
sage: SmoothCharacterGroupUnramifiedQuadratic(2, QQ).subgroup_gens(1)
[s, 2]
sage: SmoothCharacterGroupUnramifiedQuadratic(2, QQ).subgroup_gens(2)
[s, 2*s + 1, -1, 2]
sage: SmoothCharacterGroupUnramifiedQuadratic(2, QQ).subgroup_gens(3)
[s, 2*s + 1, 4*s + 1, -1, 2]

unit_gens(c)
A list of generators \(x_1, \ldots, x_d\) of the abelian group \(F^\times/(1 + p^c)^\times\), where \(c\) is the given level, satisfying no relations other than \(x_i^{n_i} = 1\) for each \(i\) (where the integers \(n_i\) are returned by \texttt{exponents()}). We adopt the convention that the final generator \(x_d\) is a uniformiser (and \(n_d = 0\)).

ALGORITHM: Use Teichmueller lifts.

EXAMPLES:

sage: from sage.modular.local_comp.smoothchar import SmoothCharacterGroupUnramifiedQuadratic
sage: SmoothCharacterGroupUnramifiedQuadratic(7, QQ).unit_gens(0)
[7]
sage: SmoothCharacterGroupUnramifiedQuadratic(7, QQ).unit_gens(1)
[s, 7]
sage: SmoothCharacterGroupUnramifiedQuadratic(7, QQ).unit_gens(2)
[22*s, 8, 7*s + 1, 7]
sage: SmoothCharacterGroupUnramifiedQuadratic(7, QQ).unit_gens(3)
[169*s + 49, 8, 7*s + 1, 7]

In the 2-adic case there can be more than 4 generators:
4.7 Type spaces of newforms

Let $f$ be a new modular eigenform of level $\Gamma_1(N)$, and $p$ a prime dividing $N$, with $N = Mp^r$ ($M$ coprime to $p$). Suppose the power of $p$ dividing the conductor of the character of $f$ is $p^c$ ($0 \leq c \leq r$).

Then there is an integer $u$, which is $\min(\lceil r/2 \rceil, r - c)$, such that any twist of $f$ by a character mod $p^u$ also has level $N$. The type space of $f$ is the span of the modular eigensymbols corresponding to all of these twists, which lie in a space of modular symbols for a suitable $\Gamma_H$ subgroup. This space is the key to computing the isomorphism class of the local component of the newform at $p$.

```python
class sage.modular.local_comp.type_space.TypeSpace(f, p, base_extend=True)
    Bases: SageObject
    The modular symbol type space associated to a newform, at a prime dividing the level.

class sage.modular.local_comp.type_space.TypeSpace(f, p, base_extend=True)
    character_conductor()
        Exponent of $p$ dividing the conductor of the character of the form.

        EXAMPLES:
        sage: from sage.modular.local_comp.type_space import example_type_space
        sage: example_type_space().character_conductor()
        0

class sage.modular.local_comp.type_space.TypeSpace(f, p, base_extend=True)
    conductor()
        Exponent of $p$ dividing the level of the form.

        EXAMPLES:
        sage: from sage.modular.local_comp.type_space import example_type_space
        sage: example_type_space().conductor()
        2

class sage.modular.local_comp.type_space.TypeSpace(f, p, base_extend=True)
    eigensymbol_subspace()
        Return the subspace of self corresponding to the plus eigensymbols of $f$ and its Galois conjugates (as a subspace of the vector space returned by $free_module()$).

        EXAMPLES:
        sage: from sage.modular.local_comp.type_space import example_type_space
        sage: T = example_type_space(); T.eigensymbol_subspace()
        Vector space of degree 6 and dimension 1 over Number Field in a1 with defining polynomial ...
        Basis matrix:
        [...]    sage: T.eigensymbol_subspace().is_submodule(T.free_module())
        True

class sage.modular.local_comp.type_space.TypeSpace(f, p, base_extend=True)
    form()
        The newform of which this is the type space.

        EXAMPLES:
        sage: from sage.modular.local_comp.type_space import example_type_space
        sage: example_type_space().form()
        q + ... + O(q^6)
```
**free_module()**

Return the underlying vector space of this type space.

EXAMPLES:

```python
sage: from sage.modular.local_comp.type_space import example_type_space
sage: example_type_space().free_module()
Vector space of dimension 6 over Number Field in a1 with defining polynomial ...
```

**group()**

Return a \( \Gamma_H \) group which is the level of all of the relevant twists of \( f \).

EXAMPLES:

```python
sage: from sage.modular.local_comp.type_space import example_type_space
sage: example_type_space().group()
Congruence Subgroup Gamma_H(98) with H generated by [15, 29, 43]
```

**is_minimal()**

Return True if there exists a newform \( g \) of level strictly smaller than \( N \), and a Dirichlet character \( \chi \) of \( p \)-power conductor, such that \( f = g \otimes \chi \) where \( f \) is the form of which this is the type space. To find such a form, use \texttt{minimal_twist()}.

The result is cached.

EXAMPLES:

```python
sage: from sage.modular.local_comp.type_space import example_type_space
sage: example_type_space().is_minimal()
True
sage: example_type_space(1).is_minimal()
False
```

**minimal_twist()**

Return a newform (not necessarily unique) which is a twist of the original form \( f \) by a Dirichlet character of \( p \)-power conductor, and which has minimal level among such twists of \( f \).

An error will be raised if \( f \) is already minimal.

EXAMPLES:

```python
sage: from sage.modular.local_comp.type_space import TypeSpace, example_type_space
sage: T = example_type_space(1)
sage: T.form().q_expansion(12)
q - q^2 + 2*q^3 + q^4 - 2*q^6 - q^8 + q^9 + O(q^12)
sage: g = T.minimal_twist()
sage: g.q_expansion(12)
q - q^2 - 2*q^3 + q^4 + 2*q^6 + q^7 - q^8 + q^9 + O(q^12)
sage: g.level()
14
sage: TypeSpace(g, 7).is_minimal()
True
```

Test that github issue #13158 is fixed:
```
sage: f = Newforms(256,names='a')[0]
sage: T = TypeSpace(f,2) # long time
sage: g = T.minimal_twist() # long time
sage: g[0:3] # long time
[0, 1, 0]
sage: str(g[3]) in ('a', '-a', '-1/2*a', '1/2*a') # long time
True
sage: g[4:] # long time
[]
sage: g.level() # long time
64
```

prime()  
Return the prime \(p\).

EXAMPLES:
```
sage: from sage.modular.local_comp.type_space import example_type_space
sage: example_type_space().prime()
7
```

rho(g)  
Calculate the action of the group element \(g\) on the type space.

EXAMPLES:
```
sage: from sage.modular.local_comp.type_space import example_type_space
sage: T = example_type_space(2)
sage: m = T.rho([2,0,0,1]); m
[-1  1  0 -1]
[ 0  0 -1  1]
[ 0 -1 -1  1]
[ 1 -1 -2  2]
sage: v = T.eigensymbol_subspace().basis()[0]
sage: m * v == v
True
```

We test that it is a left action:
```
sage: T = example_type_space(0)
sage: a = [0,5,4,3]; b = [0,2,3,5]; ab = [1,4,2,2]
sage: T.rho(ab) == T.rho(a) * T.rho(b)
True
```

An odd level example:
```
sage: from sage.modular.local_comp.type_space import TypeSpace
sage: T = TypeSpace(Newform('54a'), 3)
sage: a = [0,1,3,0]; b = [2,1,0,1]; ab = [0,1,6,3]
sage: T.rho(ab) == T.rho(a) * T.rho(b)
True
```

tame_level()  
The level away from \(p\).

4.7. Type spaces of newforms 363
EXAMPLES:

```python
sage: from sage.modular.local_comp.type_space import example_type_space
sage: example_type_space().tame_level()
2
```

$u()$

Largest integer $u$ such that level of $f_\chi = \text{level of } f$ for all Dirichlet characters $\chi$ modulo $p^u$.

EXAMPLES:

```python
sage: from sage.modular.local_comp.type_space import example_type_space
sage: example_type_space().u()
1
sage: from sage.modular.local_comp.type_space import TypeSpace
sage: f = Newforms(Gamma1(5), 5, names='a')[0]
sage: TypeSpace(f, 5).u()
0
```

`sage.modular.local_comp.type_space.example_type_space()`

Quickly return an example of a type space. Used mainly to speed up doctesting.

EXAMPLES:

```python
sage: from sage.modular.local_comp.type_space import example_type_space
sage: example_type_space()  # takes a while but caches stuff (21s on sage.math, →2012)
6-dimensional type space at prime 7 of form q + ... + O(q^6)
```

The above test takes a long time, but it precomputes and caches various things such that subsequent doctests can be very quick. So we don’t want to mark it `# long time`.

`sage.modular.local_comp.type_space.find_in_space(f, A, base_extend=False)`

Given a Newform object $f$, and a space $A$ of modular symbols of the same weight and level, find the subspace of $A$ which corresponds to the Hecke eigenvalues of $f$.

If `base_extend = True`, this will return a 2-dimensional space generated by the plus and minus eigensymbols of $f$. If `base_extend = False` it will return a larger space spanned by the eigensymbols of $f$ and its Galois conjugates.

(NB: “Galois conjugates” needs to be interpreted carefully – see the last example below.)

$A$ should be an ambient space (because non-ambient spaces don’t implement `base_extend`).

EXAMPLES:

```python
sage: from sage.modular.local_comp.type_space import find_in_space
```

Easy case ($f$ has rational coefficients):

```python
sage: f = Newform('99a'); f
q - q^2 - q^4 - 4*q^5 + O(q^6)
sage: A = ModularSymbols(GammaH(99, [13]))
sage: find_in_space(f, A)
Modular Symbols subspace of dimension 2 of Modular Symbols space of dimension 25 for Congruence Subgroup Gamma_H(99) with H generated by [13] of weight 2 with sign 0 over Rational Field
```
Harder case:

```python
sage: f = Newforms(23, names='a')[0]
sage: A = ModularSymbols(Gamma1(23))
sage: find_in_space(f, A, base_extend=True)
Modular Symbols subspace of dimension 2 of Modular Symbols space of dimension 45
→ for Gamma_1(23) of weight 2 with sign 0 over Number Field in a0 with defining...
→ polynomial x^2 + x - 1
sage: find_in_space(f, A, base_extend=False)
Modular Symbols subspace of dimension 4 of Modular Symbols space of dimension 45
→ for Gamma_1(23) of weight 2 with sign 0 over Rational Field
```

An example with character, indicating the rather subtle behaviour of base_extend:

```python
sage: chi = DirichletGroup(5).0
sage: f = Newforms(chi, 7, names='c')[0]; f  # long time (4s on sage.math, 2012)
q + c0*q^2 + (zeta4*c0 - 5*zeta4 + 5)*q^3 + ((-5*zeta4 - 5)*c0 + 24*zeta4)*q^4 +
→ ((10*zeta4 - 5)*c0 - 40*zeta4 - 55)*q^5 + O(q^6)
sage: find_in_space(f, ModularSymbols(Gamma1(5), 7), base_extend=True)  # long time
Modular Symbols subspace of dimension 2 of Modular Symbols space of dimension 12
→ for Gamma_1(5) of weight 7 with sign 0 over Number Field in c0 with defining...
→ polynomial x^2 + (5*zeta4 + 5)*x - 88*zeta4 over its base field
sage: find_in_space(f, ModularSymbols(Gamma1(5), 7), base_extend=False)  # long...
→ time (27s on sage.math, 2012)
Modular Symbols subspace of dimension 4 of Modular Symbols space of dimension 12
→ for Gamma_1(5) of weight 7 with sign 0 over Cyclotomic Field of order 4 and...
→ degree 2
```

Note that the base ring in the second example is \(\mathbb{Q}(\zeta_4)\) (the base ring of the character of \(f\)), not \(\mathbb{Q}\).

### 4.8 Helper functions for local components

This module contains various functions relating to lifting elements of \(SL_2(\mathbb{Z}/N\mathbb{Z})\) to \(SL_2(\mathbb{Z})\), and other related problems.

**sage.modular.local_comp.liftings.lift_for_SL(A, N=None)**

Lift a matrix \(A\) from \(SL_m(\mathbb{Z}/N\mathbb{Z})\) to \(SL_m(\mathbb{Z})\).

This follows [Shi1971], Lemma 1.38, p. 21.

**INPUT:**

- \(A\) – a square matrix with coefficients in \(\mathbb{Z}/N\mathbb{Z}\) (or \(\mathbb{Z}\))
- \(N\) – the modulus (optional) required only if the matrix \(A\) has coefficients in \(\mathbb{Z}\)

**EXAMPLES:**

```python
sage: from sage.modular.local_comp.liftings import lift_for_SL
sage: A = matrix(Zmod(11), 4, 4, [6, 0, 0, 9, 1, 6, 9, 4, 4, 4, 8, 0, 4, 0, 0, 8])
sage: A.det()  
1
sage: L = lift_for_SL(A)
sage: L.det()  
1
```

(continues on next page)
sage: (L - A) == 0
True

sage: B = matrix(Zmod(19), 4, 4, [1, 6, 10, 4, 4, 14, 15, 4, 13, 0, 1, 15, 15, 0, 
→ 17, 10])
sage: B.det()
1
sage: L = lift_for_SL(B)
sage: L.det()
1
sage: (L - B) == 0
True

sage.modular.local_comp.liftings.lift_gen_to_gamma1(m, n)

Return four integers defining a matrix in \( SL_2(\mathbb{Z}) \) which is congruent to \( \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} \) (mod \( m \)) and lies in the subgroup \( \begin{pmatrix} 1 & * \\ 0 & 1 \end{pmatrix} \) (mod \( n \)).

This is a special case of \( lift_to_gamma1() \), and is coded as such.

INPUT:

* \( m, n \) – coprime positive integers

EXAMPLES:

sage: from sage.modular.local_comp.liftings import lift_gen_to_gamma1
sage: A = matrix(ZZ, 2, lift_gen_to_gamma1(9, 8)); A
\[
\begin{bmatrix}
441 & 62 \\
64 & 9
\end{bmatrix}
\]
sage: A.change_ring(Zmod(9))
\[
\begin{bmatrix}
0 & 8 \\
1 & 0
\end{bmatrix}
\]
sage: A.change_ring(Zmod(8))
\[
\begin{bmatrix}
1 & 6 \\
0 & 1
\end{bmatrix}
\]
sage: type(lift_gen_to_gamma1(9, 8)[0])
<class 'sage.rings.integer.Integer'>

sage.modular.local_comp.liftings.lift_matrix_to_sl2z(A, N)

Given a list of length 4 representing a 2x2 matrix over \( \mathbb{Z}/N\mathbb{Z} \) with determinant 1 (mod \( N \), lift it to a 2x2 matrix over \( \mathbb{Z} \) with determinant 1.

This is a special case of \( lift_to_gamma1() \), and is coded as such.

INPUT:

* \( A \) – list of 4 integers defining a \( 2 \times 2 \) matrix
* \( N \) – positive integer

EXAMPLES:

sage: from sage.modular.local_comp.liftings import lift_matrix_to_sl2z
sage: lift_matrix_to_sl2z([10, 11, 3, 11], 19)
sage: type(_[0])
<class 'sage.rings.integer.Integer'>

sage: lift_matrix_to_sl2z([2,0,0,1], 5)
Traceback (most recent call last):
  ...
ValueError: Determinant is 2 mod 5, should be 1

sage.modular.local_comp.liftings.lift_ramified(g, p, u, n)

Given four integers \(a, b, c, d\) with \(p \mid c\) and \(ad - bc = 1 \pmod{p^u}\), find \(a', b', c', d'\) congruent to \(a, b, c, d\) \((\pmod{p^u})\), with \(c' = c \pmod{p^{u+1}}\), such that \(a'd' - b'c'\) is exactly 1, and \(\begin{pmatrix} a & b \\ c & d \end{pmatrix}\) is in \(\Gamma_1(n)\).

Algorithm: Uses \(\text{lift_to_gamma1()}\) to get a lifting modulo \(p^u\), and then adds an appropriate multiple of the top row to the bottom row in order to get the bottom-left entry correct modulo \(p^{u+1}\).

EXAMPLES:

sage: from sage.modular.local_comp.liftings import lift_ramified
sage: lift_ramified([2,2,3,2], 3, 1, 1)
[-1, -1, 3, 2]
sage: lift_ramified([8,2,12,2], 3, 2, 23)
[323, 110, -133584, -45493]
sage: type(lift_ramified([8,2,12,2], 3, 2, 23)[0])
<class 'sage.rings.integer.Integer'>

sage.modular.local_comp.liftings.lift_to_gamma1(g, m, n)

If \(g = [a, b, c, d]\) is a list of integers defining a \(2 \times 2\) matrix whose determinant is \(1 \pmod{m}\), return a list of integers giving the entries of a matrix which is congruent to \(g \pmod{m}\) and to \(\begin{pmatrix} 1 & * \\ 0 & 1 \end{pmatrix} \pmod{n}\). Here \(m\) and \(n\) must be coprime.

INPUT:

- \(g\) – list of 4 integers defining a \(2 \times 2\) matrix
- \(m, n\) – coprime positive integers

Here \(m\) and \(n\) should be coprime positive integers. Either of \(m\) and \(n\) can be 1. If \(n = 1\), this still makes perfect sense; this is what is called by the function \(\text{lift_matrix_to_sl2z()}\). If \(m = 1\) this is a rather silly question, so we adopt the convention of always returning the identity matrix.

The result is always a list of Sage integers (unlike \(\text{lift_to_sl2z}\), which tends to return Python ints).

EXAMPLES:

sage: from sage.modular.local_comp.liftings import lift_to_gamma1
sage: A = matrix(ZZ, 2, lift_to_gamma1([10, 11, 3, 11], 19, 5)); A
[371 68]
[60 11]
sage: A.det() == 1
True
sage: A.change_ring(Zmod(19))
[10 11]
[3 11]
sage: A.change_ring(Zmod(5))
sage: m = list(SL2Z.random_element())
sage: n = lift_to_gamma1(m, 11, 17)
sage: assert matrix(Zmod(11), 2, n) == matrix(Zmod(11), 2, m)
sage: assert matrix(Zmod(17), 2, [n[0], 0, n[2], n[3]]) == 1
sage: type(lift_to_gamma1([10,11,3,11],19,5)[0])
<class 'sage.rings.integer.Integer'>

Tests with $m = 1$ and with $n = 1$:

```
sage: lift_to_gamma1([1,1,0,1], 5, 1)
[1, 1, 0, 1]
sage: lift_to_gamma1([2,3,11,22], 1, 5)
[1, 0, 0, 1]
```

```
sage.modular.local_comp.liftings.lift_uniformiser_odd(p, u, n)

Construct a matrix over $\mathbb{Z}$ whose determinant is $p$, and which is congruent to
$\begin{pmatrix} 0 & -1 \\ p & 0 \end{pmatrix}$ (mod $p^n$) and to
$\begin{pmatrix} p & 0 \\ 0 & 1 \end{pmatrix}$ (mod $n$).

This is required for the local components machinery in the “ramified” case (when the exponent of $p$ dividing the level is odd).

EXAMPLES:
```
sage: from sage.modular.local_comp.liftings import lift_uniformiser_odd
sage: lift_uniformiser_odd(3, 2, 11)
[432, 377, 165, 144]
sage: type(lift_uniformiser_odd(3, 2, 11)[0])
<class 'sage.rings.integer.Integer'>
```

4.9 Eta-products on modular curves $X_0(N)$

This package provides a class for representing eta-products, which are meromorphic functions on modular curves of the form

$$
\prod_{d|N} \eta(q^d)^{r_d}
$$

where $\eta(q)$ is Dirichlet’s eta function

$$
q^{1/24} \prod_{n=1}^{\infty} (1 - q^n).
$$

These are useful for obtaining explicit models of modular curves.

See github issue #3934 for background.

AUTHOR:

- David Loeffler (2008-08-22): initial version
sage.modular.etaproducts.AllCusps(N)
Return a list of CuspFamily objects corresponding to the cusps of $X_0(N)$.
INPUT:
• N (integer): the level
EXAMPLES:

```sage
sage: AllCusps(18)
[(Inf), (c_{2}), (c_{3,1}), (c_{3,2}), (c_{6,1}), (c_{6,2}), (c_{9}), (0)]
sage: AllCusps(0)
Traceback (most recent call last):
... ValueError: N must be positive
```

class sage.modular.etaproducts.CuspFamily(N, width, label=None)
Bases: SageObject
A family of elliptic curves parametrising a region of $X_0(N)$.

level()
Return the level of this cusp.
EXAMPLES:

```sage
e = CuspFamily(10, 1)
sage: e.level()
10
```
sage_cusp()
Return the corresponding element of $\mathbb{P}^1(\mathbb{Q})$.
EXAMPLES:

```sage
CuspFamily(10, 1).sage_cusp() # not implemented
infinity
```
width()
Return the width of this cusp.
EXAMPLES:

```sage
e = CuspFamily(10, 1)
sage: e.width()
1
```
sage.modular.etaproducts.EtaGroup(level)
Create the group of eta products of the given level.
EXAMPLES:

```sage
EtaGroup(12)
Group of eta products on $X_0(12)$
sage: EtaGroup(1/2)
Traceback (most recent call last):
... ValueError: N must be positive
```

4.9. Eta-products on modular curves $X_0(N)$
TypeError: Level (=1/2) must be a positive integer

```
sage: EtaGroup(0)
Traceback (most recent call last):
...
ValueError: Level (=0) must be a positive integer
```

**class** `sage.modular.etaproducts.EtaGroupElement(parent, rdict)`

Bases: `Element`

Create an eta product object. Usually called implicitly via `EtaGroup_class.__call__` or the `EtaProduct` factory function.

**EXAMPLES:**

```
sage: EtaProduct(8, {1:24, 2:-24})
Eta product of level 8 : (eta_1)^24 (eta_2)^-24
sage: g = _; g == loads(dumps(g))
True
sage: TestSuite(g).run()
```

**degree()**

Return the degree of `self` as a map $X_0(N) \to \mathbb{P}^1$.

This is the sum of all the positive coefficients in the divisor of `self`.

**EXAMPLES:**

```
sage: e = EtaProduct(12, {1:-336, 2:576, 3:696, 4:-216, 6:-576, 12:-144})
sage: e.degree()
230
```

**divisor()**

Return the divisor of `self`, as a formal sum of `CuspFamily` objects.

**EXAMPLES:**

```
sage: e = EtaProduct(12, {1:-336, 2:576, 3:696, 4:-216, 6:-576, 12:-144})
sage: e.divisor() # random
-131*(Inf) - 50*(c_{2}) + 11*(0) + 50*(c_{6}) + 169*(c_{4}) - 49*(c_{3})
sage: e = EtaProduct(2^8, {8:1,32:-1})
sage: e.divisor() # random
-(c_{2}) - (Inf) - (c_{8,2}) - (c_{8,3}) - (c_{8,4}) - (c_{4,2})
- (c_{8,1}) - (c_{4,1}) + (c_{32,4}) + (c_{32,3}) + (c_{64,1})
+ (0) + (c_{32,2}) + (c_{64,2}) + (c_{128}) + (c_{32,1})
```

**is_one()**

Return whether `self` is the one of the monoid.

**EXAMPLES:**

```
sage: e = EtaProduct(3, {3:12, 1:-12})
sage: e.is_one()
False
sage: e.parent().one().is_one()
True
```
sage: ep = EtaProduct(5, {})  
sage: ep.is_one()  
True  
sage: ep.parent().one() == ep  
True

level()

Return the level of this eta product.

EXAMPLES:

sage: e = EtaProduct(3, {3:12, 1:-12})  
sage: e.level()  
3  
sage: EtaProduct(12, {6:6, 2:-6}).level() # not the lcm of the d's  
12  
sage: EtaProduct(36, {6:6, 2:-6}).level() # not minimal  
36

order_at_cusp(cusp)

Return the order of vanishing of self at the given cusp.

INPUT:

• cusp -- a CuspFamily object

OUTPUT:

• an integer

EXAMPLES:

sage: e = EtaProduct(2, {2:24, 1:-24})  
sage: e.order_at_cusp(CuspFamily(2, 1)) # cusp at infinity  
1  
sage: e.order_at_cusp(CuspFamily(2, 2)) # cusp 0  
-1

q_expansion(n)

Return the q-expansion of self at the cusp at infinity.

INPUT:

• n (integer): number of terms to calculate

OUTPUT:

• a power series over \( \mathbb{Z} \) in the variable \( q \), with a relative precision of \( 1 + O(q^n) \).

ALGORITHM: Calculates \( \eta \) to \( (n/m) \) terms, where \( m \) is the smallest integer dividing self.level() such that \( self.r(m) \neq 0 \). Then multiplies.

EXAMPLES:

sage: EtaProduct(36, {6:6, 2:-6}).q_expansion(10)  
q + 6*q^3 + 27*q^5 + 92*q^7 + 279*q^9 + O(q^11)  
sage: R.<q> = ZZ[[q]]  
sage: EtaProduct(2,{2:24,1:-24}).q_expansion(100) == delta_qexp(101)(q^2)/delta_
qexp(n)

Alias for self.q_expansion().

EXAMPLES:

```python
sage: e = EtaProduct(36, {6:8, 3:-8})
sage: e.qexp(10)
q + 8*q^4 + 36*q^7 + O(q^10)
sage: e.qexp(30) == e.q_expansion(30)
True
```

r(d)

Return the exponent \( r_d \) of \( \eta(q^d) \) in self.

EXAMPLES:

```python
sage: e = EtaProduct(12, {2:24, 3:-24})
sage: e.r(3)
-24
sage: e.r(4)
0
```

class sage.modular.etaproducts.EtaGroup_class(level)

Bases: UniqueRepresentation, Parent

The group of eta products of a given level under multiplication.

Element

alias of EtaGroupElement

basis(reduce=True)

Produce a basis for the free abelian group of eta-products of level \( N \) (under multiplication), attempting to find basis vectors of the smallest possible degree.

INPUT:

- reduce - a boolean (default True) indicating whether or not to apply LLL-reduction to the calculated basis

EXAMPLES:

```python
sage: EtaGroup(5).basis()
[\text{Eta product of level 5 : (eta_1)^6 (eta_5)^{-6}}]
sage: EtaGroup(12).basis()
[\text{Eta product of level 12 : (eta_1)^{-3} (eta_2)^2 (eta_3)^1 (eta_4)^{-1} (eta_6)^{-2} \rightarrow (eta_12)^3},
\text{Eta product of level 12 : (eta_1)^{-4} (eta_2)^2 (eta_3)^4 (eta_6)^{-2},}
\text{Eta product of level 12 : (eta_1)^6 (eta_2)^{-9} (eta_3)^{-2} (eta_4)^3 (eta_6)^3 \rightarrow (eta_12)^{-1},}
\text{Eta product of level 12 : (eta_1)^{-1} (eta_2)^3 (eta_3)^3 (eta_4)^{-2} (eta_6)^{-9} \rightarrow (eta_12)^6,}
\text{Eta product of level 12 : (eta_1)^3 (eta_3)^{-1} (eta_4)^{-3} (eta_12)^1}]
```
sage: EtaGroup(12).basis(reduce=False) # much bigger coefficients
[Eta product of level 12 : (eta_1)^384 (eta_2)^-576 (eta_3)^-696 (eta_4)^216,
 →(eta_6)^576 (eta_12)^96,
 Eta product of level 12 : (eta_2)^24 (eta_12)^-24,
 Eta product of level 12 : (eta_1)^-40 (eta_2)^116 (eta_3)^96 (eta_4)^-30 (eta_ →6)^-80 (eta_12)^-62,
 Eta product of level 12 : (eta_1)^-4 (eta_2)^-33 (eta_3)^-4 (eta_4)^1 (eta_6)^ →3 (eta_12)^37,
 Eta product of level 12 : (eta_1)^15 (eta_2)^-24 (eta_3)^-29 (eta_4)^9 (eta_6)^ →24 (eta_12)^5]

**ALGORITHM:** An eta product of level \( N \) is uniquely determined by the integers \( r_d \) for \( d | N \) with \( d < N \), since \( \sum_{d | N} r_d = 0 \). The valid \( r_d \) are those that satisfy two congruences modulo 24, and one congruence modulo 2 for every prime divisor of \( N \). We beef up the congruences modulo 2 to congruences modulo 24 by multiplying by 12. To calculate the kernel of the ensuing map \( \mathbb{Z}^m \to (\mathbb{Z}/24\mathbb{Z})^n \) we lift it arbitrarily to an integer matrix and calculate its Smith normal form. This gives a basis for the lattice.

This lattice typically contains “large” elements, so by default we pass it to the reduce_basis() function which performs LLL-reduction to give a more manageable basis.

**level()**

Return the level of \( self \).

**EXAMPLES:**

sage: EtaGroup(10).level()
10

**one()**

Return the identity element of \( self \).

**EXAMPLES:**

sage: EtaGroup(12).one()
Eta product of level 12 : 1

**reduce_basis**(long_etas)

Produce a more manageable basis via LLL-reduction.

**INPUT:**

- long_etas - a list of EtaGroupElement objects (which should all be of the same level)

**OUTPUT:**

- a new list of EtaGroupElement objects having hopefully smaller norm

**ALGORITHM:** We define the norm of an eta-product to be the \( L^2 \) norm of its divisor (as an element of the free \( \mathbb{Z} \)-module with the cusps as basis and the standard inner product). Applying LLL-reduction to this gives a basis of hopefully more tractable elements. Of course we’d like to use the \( L^1 \) norm as this is just twice the degree, which is a much more natural invariant, but \( L^2 \) norm is easier to work with!

**EXAMPLES:**

sage: EtaGroup(4).reduce_basis([ EtaProduct(4, {1:8,2:24,4:-32}), EtaProduct(4, →{1:8, 4:-8})])
sage.modular.etaproducts.EtaProduct(level, dic)

Create an EtaGroupElement object representing the function \( \prod_{d \mid N} \eta(q^d)^{r_d} \).

This checks the criteria of Ligozat to ensure that this product really is the \( q \)-expansion of a meromorphic function on \( X_0(N) \).

**INPUT:**
- **level** – (integer): the \( N \) such that this eta product is a function on \( X_0(N) \).
- **dic** – (dictionary): a dictionary indexed by divisors of \( N \) such that the coefficient of \( \eta(q^d) \) is \( r_d \). Only nonzero coefficients need be specified. If Ligozat’s criteria are not satisfied, a `ValueError` will be raised.

**OUTPUT:**
- an EtaGroupElement object, whose parent is the EtaGroup of level \( N \) and whose coefficients are the given dictionary.

**Note:** The dictionary `dic` does not uniquely specify \( N \). It is possible for two EtaGroupElements with different \( N \)’s to be created with the same dictionary, and these represent different objects (although they will have the same \( q \)-expansion at the cusp \( \infty \)).

**EXAMPLES:**

```
sage: EtaProduct(3, {3:12, 1:-12})
Eta product of level 3 : (eta_1)^-12 (eta_3)^12
sage: EtaProduct(3, {3:6, 1:-6})
Traceback (most recent call last):
  ... ValueError: sum d r_d (=12) is not 0 mod 24
sage: EtaProduct(3, {4:6, 1:-6})
Traceback (most recent call last):
  ... ValueError: 4 does not divide 3
```

sage.modular.etaproducts.eta_poly_relations(eta_elements, degree, labels=['x1', 'x2'], verbose=False)

Find polynomial relations between eta products.

**INPUT:**
- **eta_elements** - (list): a list of EtaGroupElement objects. Not implemented unless this list has precisely two elements. degree
- **degree** - (integer): the maximal degree of polynomial to look for.
- **labels** - (list of strings): labels to use for the polynomial returned.
- **verbose** - (boolean, default False): if True, prints information as it goes.

**OUTPUT:** a list of polynomials which is a Groebner basis for the part of the ideal of relations between eta_elements which is generated by elements up to the given degree; or None, if no relations were found.

**ALGORITHM:** An expression of the form \( \sum_{0 \leq i,j \leq d} a_{ij} x^i y^j \) is zero if and only if it vanishes at the cusp infinity to degree at least \( v = d(deg(x) + deg(y)) \). For all terms up to \( q^v \) in the \( q \)-expansion of this expression to be zero
is a system of $v + k$ linear equations in $d^2$ coefficients, where $k$ is the number of nonzero negative coefficients that can appear.

Solving these equations and calculating a basis for the solution space gives us a set of polynomial relations, but this is generally far from a minimal generating set for the ideal, so we calculate a Groebner basis.

As a test, we calculate five extra terms of $q$-expansion and check that this doesn’t change the answer.

EXAMPLES:

```python
sage: from sage.modular.etaproducts import eta_poly_relations
sage: t = EtaProduct(26, {2:2,13:2,26:-2,1:-2})
sage: u = EtaProduct(26, {2:4,13:2,26:-4,1:-2})
sage: eta_poly_relations([t, u], 3)
sage: eta_poly_relations([t, u], 4)
```

Use `verbose=True` to see the details of the computation:

```python
sage: eta_poly_relations([t, u], 3, verbose=True)
Trying to find a relation of degree 3
Lowest order of a term at infinity = -12
Highest possible degree of a term = 15
Trying all coefficients from $q^{-12}$ to $q^{15}$ inclusive
No polynomial relation of order 3 valid for 28 terms
Check:
Trying all coefficients from $q^{-12}$ to $q^{20}$ inclusive
No polynomial relation of order 3 valid for 33 terms

sage: eta_poly_relations([t, u], 4, verbose=True)
Trying to find a relation of degree 4
Lowest order of a term at infinity = -16
Highest possible degree of a term = 20
Trying all coefficients from $q^{-16}$ to $q^{20}$ inclusive
Check:
Trying all coefficients from $q^{-16}$ to $q^{25}$ inclusive
[x1^3*x2 - 13*x1^3 - 4*x1^2*x2 - 4*x1*x2 - x2^2 + x2]
```

`sage.modular.etaproducts.num_cusps_of_width(N, d)`

Return the number of cusps on $X_0(N)$ of width $d$.

INPUT:

- $N$ – (integer): the level
- $d$ – (integer): an integer dividing $N$, the cusp width

EXAMPLES:

```python
sage: from sage.modular.etaproducts import num_cusps_of_width
sage: [num_cusps_of_width(18,d) for d in divisors(18)]
[1, 1, 2, 2, 1, 1]
sage: num_cusps_of_width(4,8)
```

Traceback (most recent call last):
...
ValueError: N and d must be positive integers with d|N
sage.modular.etaproducts.qexp_eta(ps_ring, prec)

Return the q-expansion of $\eta(q)/q^{1/24}$.

Here $\eta(q)$ is Dedekind’s function

$$\eta(q) = q^{1/24} \prod_{n=1}^{\infty} (1 - q^n).$$

The result is an element of $\text{ps}\_\text{ring}$, with precision $\text{prec}$.

INPUT:

• $\text{ps}\_\text{ring}$ – (PowerSeriesRing): a power series ring
• $\text{prec}$ – (integer): the number of terms to compute

OUTPUT: An element of $\text{ps}\_\text{ring}$ which is the q-expansion of $\eta(q)/q^{1/24}$ truncated to $\text{prec}$ terms.

ALGORITHM: We use the Euler identity

$$\eta(q) = q^{1/24} \sum_{n\geq 1} (-1)^n (q^{n(3n+1)/2} + q^{n(3n-1)/2})$$

to compute the expansion.

EXAMPLES:

```python
sage: from sage.modular.etaproducts import qexp_eta
sage: qexp_eta(ZZ[['q']], 100)
1 - q - q^2 + q^5 + q^7 - q^12 - q^15 + q^22 + q^26 - q^35 - q^40 + q^51 + q^57 - q^70 - q^77 + q^92 + O(q^100)
```

4.10 The space of $p$-adic weights

A $p$-adic weight is a continuous character $\mathbb{Z}_p^\times \to \mathbb{C}_p^\times$. These are the $\mathbb{C}_p^\times$-points of a rigid space over $\mathbb{Q}_p$, which is isomorphic to a disjoint union of copies (indexed by $(\mathbb{Z}/p\mathbb{Z})^\times$) of the open unit $p$-adic disc.

Sage supports both “classical points”, which are determined by the data of a Dirichlet character modulo $p^m$ for some $m$ and an integer $k$ (corresponding to the character $z \mapsto z^k \chi(z)$) and “non-classical points” which are determined by the data of an element of $(\mathbb{Z}/p\mathbb{Z})^\times$ and an element $w \in \mathbb{C}_p$ with $|w - 1| < 1$.

EXAMPLES:

```python
sage: W = pAdicWeightSpace(17)
sage: W
Space of 17-adic weight-characters defined over 17-adic Field with capped relative precision 20
sage: R.<x> = QQ[]
sage: L = Qp(17).extension(x^2 - 17, names='a'); L.rename('L')
sage: W.base_extend(L)
Space of 17-adic weight-characters defined over L
```

We create a simple element of $W$: the algebraic character, $x \mapsto x^6$. 
A locally algebraic character, $x \mapsto x^6 \chi(x)$ for $\chi$ a Dirichlet character mod $p$:

```sage
sage: kappa2 = W(6, DirichletGroup(17, Qp(17)).0^8)
```

```sage
sage: kappa2(5) == -5^6
True
sage: kappa2(13) == 13^6
True
```

A non-locally-algebraic character, sending the generator 18 of $1 + 17\mathbb{Z}_{17}$ to 35 and acting as $\mu \mapsto \mu^4$ on the group of 16th roots of unity:

```sage
sage: kappa3 = W(35 + O(17^20), 4, algebraic=False)
```

```sage
sage: kappa3(2)
16 + 8*17 + ... + O(17^20)
```

AUTHORS:

- David Loeffler (2008-9)

class `sage.modular.overconvergent.weightspace.AlgebraicWeight`(parent, k, chi=None)

Bases: `WeightCharacter`

A point in weight space corresponding to a locally algebraic character, of the form $x \mapsto \chi(x)x^k$ where $k$ is an integer and $\chi$ is a Dirichlet character modulo $p^n$ for some $n$.

`Lvalue()`

Return the value of the p-adic $L$-function of $\mathbb{Q}$ evaluated at this weight-character.

If the character is $x \mapsto x^k \chi(x)$ where $k > 0$ and $\chi$ has conductor a power of $p$, this is an element of the number field generated by the values of $\chi$, equal to the value of the complex $L$-function $L(1-k, \chi)$. If $\chi$ is trivial, it is equal to $(1 - p^{k-1})\zeta(1-k)$.

At present this is not implemented in any other cases, except the trivial character (for which the value is $\infty$).

Todo: Implement this more generally using the Amice transform machinery in `sage/schemes/elliptic_curves/padic_lseries.py`, which should clearly be factored out into a separate class.

EXAMPLES:

```sage
sage: pAdicWeightSpace(7)(4).Lvalue() == (1 - 7^3)*zeta__exact(-3)
True
```

```sage
sage: pAdicWeightSpace(7)(5, DirichletGroup(7, Qp(7)).0^4).Lvalue()  # 0
```

```sage
sage: pAdicWeightSpace(7)(6, DirichletGroup(7, Qp(7)).0^4).Lvalue()
1 + 2*7 + 7^2 + 3*7^3 + 4*7^5 + 2*7^7 + 5*7^8 + 2*7^9 + 3*7^10 + 6*7^11...
˓→+ 2*7^12 + 3*7^13 + 5*7^14 + 6*7^15 + 5*7^16 + 3*7^17 + 6*7^18 + 0(7^19)
```

4.10. The space of $p$-adic weights
chi()

If this character is $x \mapsto x^k \chi(x)$ for an integer $k$ and a Dirichlet character $\chi$, return $\chi$.

**EXAMPLES:**

```
sage: kappa = pAdicWeightSpace(29)(13, DirichletGroup(29, Qp(29)).0^14)
sage: kappa.chi()
Dirichlet character modulo 29 of conductor 29 mapping 2 |--> 28 + 28*29 + 28*29^2 + ... + O(29^20)
```

k()

If this character is $x \mapsto x^k \chi(x)$ for an integer $k$ and a Dirichlet character $\chi$, return $k$.

**EXAMPLES:**

```
sage: kappa = pAdicWeightSpace(29)(13, DirichletGroup(29, Qp(29)).0^14)
sage: kappa.k()
13
```

techmuller_type()

Return the Teichmuller type of this weight-character $\kappa$.

This is the unique $t \in \mathbb{Z}/(p - 1)\mathbb{Z}$ such that $\kappa(\mu) = \mu^t$ for $\mu$ a $(p - 1)$-st root of 1.

For $p = 2$ this does not make sense, but we still want the Teichmuller type to correspond to the index of the component of weight space in which $\kappa$ lies, so we return 1 if $\kappa$ is odd and 0 otherwise.

**EXAMPLES:**

```
sage: pAdicWeightSpace(11)(2, DirichletGroup(11,QQ).0).teichmuller_type()
7
sage: pAdicWeightSpace(29)(13, DirichletGroup(29, Qp(29)).0).teichmuller_type()
14
sage: pAdicWeightSpace(2)(3, DirichletGroup(4,QQ).0).teichmuller_type()
0
```

class sage.modular.overconvergent.weightspace.ArbitraryWeight(parent, w, t)

Bases: WeightCharacter

Create the element of p-adic weight space in the given component mapping $1 + p$ to $w$.

Here $w$ must be an element of a p-adic field, with finite precision.

**EXAMPLES:**

```
sage: pAdicWeightSpace(17)(1 + 17^2 + O(17^3), 11, False)
[1 + 17^2 + O(17^3), 11]
```

techmuller_type()

Return the Teichmuller type of this weight-character $\kappa$.

This is the unique $t \in \mathbb{Z}/(p - 1)\mathbb{Z}$ such that $\kappa(\mu) = \mu^t$ for $\mu$ a $(p - 1)$-st root of 1.

For $p = 2$ this does not make sense, but we still want the Teichmuller type to correspond to the index of the component of weight space in which $\kappa$ lies, so we return 1 if $\kappa$ is odd and 0 otherwise.

**EXAMPLES:**
class sage.modular.overconvergent.weightspace.WeightCharacter(parent)

Bases: Element

Abstract base class representing an element of the p-adic weight space $\text{Hom}(\mathbb{Z}_p^\times, \mathbb{C}_p^\times)$.

Lvalue()

Return the value of the p-adic L-function of $\mathbb{Q}$, which can be regarded as a rigid-analytic function on weight space, evaluated at this character.

EXAMPLES:

```
sage: W = pAdicWeightSpace(11)
sage: sage.modular.overconvergent.weightspace.WeightCharacter(W).Lvalue()
Traceback (most recent call last):
...
NotImplementedError
```

base_extend(R)

Extend scalars to the base ring R.

The ring R must have a canonical map from the current base ring.

EXAMPLES:

```
sage: w = pAdicWeightSpace(17, QQ)(3)
sage: w.base_extend(Qp(17))
3
```

is_even()

Return True if this weight-character sends -1 to +1.

EXAMPLES:

```
sage: pAdicWeightSpace(17)(0).is_even()
True
sage: pAdicWeightSpace(17)(11).is_even()
False
sage: pAdicWeightSpace(17)(1 + 17 + O(17^20), 3, False).is_even()
False
sage: pAdicWeightSpace(17)(1 + 17 + O(17^20), 4, False).is_even()
True
```

is_trivial()

Return True if and only if this is the trivial character.

EXAMPLES:

```
sage: pAdicWeightSpace(11)(2).is_trivial()
False
sage: pAdicWeightSpace(11)(2, DirichletGroup(11, QQ)(0)).is_trivial()
```
False

```sage```
pAdicWeightSpace(11)(0).is_trivial()
```
```
True

**one_over_Lvalue()**

Return the reciprocal of the p-adic L-function evaluated at this weight-character.

If the weight-character is odd, then the L-function is zero, so an error will be raised.

**EXAMPLES:**

```sage```
pAdicWeightSpace(11)(4).one_over_Lvalue()
```
```
-12/133

```sage```
pAdicWeightSpace(11)(3, DirichletGroup(11, QQ).0).one_over_Lvalue()
```
```
-1/6

```sage```
pAdicWeightSpace(11)(3).one_over_Lvalue()
```
```
Traceback (most recent call last):
...
ZeroDivisionError: rational division by zero

```sage```
pAdicWeightSpace(11)(0).one_over_Lvalue()
```
```
0

```sage```
type(_)
```
```
<class 'sage.rings.integer.Integer'>

**pAdicEisensteinSeries**(ring, prec=20)

Calculate the q-expansion of the p-adic Eisenstein series of given weight-character, normalised so the constant term is 1.

**EXAMPLES:**

```sage```
kappa = pAdicWeightSpace(3)(3, DirichletGroup(3, QQ).0)
kappa.pAdicEisensteinSeries(QQ[['q']], 20)
```
```
1 - 9*q + 27*q^2 - 9*q^3 - 117*q^4 + 216*q^5 + 27*q^6 - 450*q^7 + 459*q^8 - 9*q^9 - 648*q^10 + 1080*q^11 - 117*q^12 - 1530*q^13 + 1350*q^14 + 216*q^15 - 1845*q^16 + 2592*q^17 + 27*q^18 - 3258*q^19 + O(q^20)

**values_on_gens()**

If κ is this character, calculate the values \((κ(r), t)\) where \(r = 1 + p\) (or 5 if \(p = 2\)) and \(t\) is the unique element of \(\mathbb{Z}/(p - 1)\mathbb{Z}\) such that \(κ(μ) = μ^t\) for \(μ\) a (p-1)st root of unity. (If \(p = 2\), we take \(t\) to be 0 or 1 according to whether \(κ\) is odd or even.) These two values uniquely determine the character κ.

**EXAMPLES:**

```sage```
W = pAdicWeightSpace(11); W(2).values_on_gens()
```
```
(1 + 2*11^1 + 11^2 + O(11^20), 2)

```sage```
W(2, DirichletGroup(11, QQ).0).values_on_gens()
```
```
(1 + 2*11^1 + 11^2 + O(11^20), 7)

```sage```
W(1 + 2*11 + O(11^5), 4, algebraic = False).values_on_gens()
```
```
(1 + 2*11 + O(11^5), 4)

**class** sage.modular.overconvergent.weighspace.WeightSpace_class(p, base_ring)

**Bases:** Parent

The space of \(p\)-adic weight-characters \(W = \text{Hom}(\mathbb{Z}_p^\times, C_p^\times)\).
This is isomorphic to a disjoint union of \((p - 1)\) open discs of radius 1 (or 2 such discs if \(p = 2\)), with the parameter on the open disc corresponding to the image of \(1 + p\) (or 5 if \(p = 2\)).

**base extend** \((R)\)

Extend scalars to the ring \(R\).

There must be a canonical coercion map from the present base ring to \(R\).

**EXAMPLES:**

```
sage: W = pAdicWeightSpace(3, QQ)
sage: W.base_extend(Qp(3))
```

```
Space of 3-adic weight-characters defined over 3-adic Field with capped relative precision 20
```

```
sage: W.base_extend(IntegerModRing(12))
```

```
Traceback (most recent call last):
...
TypeError: No coercion map from 'Rational Field' to 'Ring of integers modulo 12' is defined
```

**prime**

Return the prime \(p\) such that this is a \(p\)-adic weight space.

**EXAMPLES:**

```
sage: pAdicWeightSpace(17).prime()
```

```
17
```

**zero**

Return the zero of this weight space.

**EXAMPLES:**

```
sage: W = pAdicWeightSpace(17)
sage: W.zero()
```

```
0
```

`sage.modular.overconvergent.weightspace.WeightSpace_constructor(p, base_ring=None)`

Construct the \(p\)-adic weight space for the given prime \(p\).

A \(p\)-adic weight is a continuous character \(\mathbb{Z}_p^\times \to \mathbb{C}_p^\times\). These are the \(\mathbb{C}_p\)-points of a rigid space over \(\mathbb{Q}_p\), which is isomorphic to a disjoint union of copies (indexed by \((\mathbb{Z}/p\mathbb{Z})^\times\)) of the open unit \(p\)-adic disc.

Note that the “base ring” of a \(p\)-adic weight is the smallest ring containing the image of \(\mathbb{Z}\); in particular, although the default base ring is \(\mathbb{Q}_p\), base ring \(\mathbb{Q}\) will also work.

**EXAMPLES:**

```
sage: pAdicWeightSpace(3) # indirect doctest
```

```
Space of 3-adic weight-characters defined over 3-adic Field with capped relative precision 20
```

```
sage: pAdicWeightSpace(3, QQ)
```

```
Space of 3-adic weight-characters defined over Rational Field
```

```
sage: pAdicWeightSpace(10)
```

```
Traceback (most recent call last):
...
ValueError: p must be prime
```
4.11 Overconvergent $p$-adic modular forms for small primes

This module implements computations of Hecke operators and $U_p$-eigenfunctions on $p$-adic overconvergent modular forms of tame level 1, where $p$ is one of the primes $\{2, 3, 5, 7, 13\}$, using the algorithms described in [Loe2007].

- [Loe2007]

AUTHORS:
- David Loeffler (August 2008): initial version
- David Loeffler (March 2009): extensively reworked
- David Loeffler (June 2009): miscellaneous bug fixes and usability improvements

4.11.1 The Theory

Let $p$ be one of the above primes, so $X_0(p)$ has genus 0, and let

$$f_p = p^{-\frac{1}{2}} \sqrt{\frac{\Delta(p^2)}{\Delta(z)}}$$

(an $\eta$-product of level $p$ – see module `sage.modular.etaproducts`). Then one can show that $f_p$ gives an isomorphism $X_0(p) \to \mathbb{P}^1$. Furthermore, if we work over $\mathbb{C}_p$, the $r$-overconvergent locus on $X_0(p)$ (or of $X_0(1)$, via the canonical subgroup lifting), corresponds to the $p$-adic disc

$$|f_p|_p \leq p^{-\frac{12r}{p-1}}.$$  

(This is Theorem 1 of [Loe2007].)

Hence if we fix an element $c$ with $|c| = p^{-\frac{12r}{p-1}}$, the space $S_k^!(1, r)$ of overconvergent $p$-adic modular forms has an orthonormal basis given by the functions $(cf)^n$. So any element can be written in the form $E_k \times \sum_{n \geq 0} a_n(cf)^n$, where $a_n \to 0$ as $N \to \infty$, and any such sequence $a_n$ defines a unique overconvergent form.

One can now find the matrix of Hecke operators in this basis, either by calculating $q$-expansions, or (for the special case of $U_p$) using a recurrence formula due to Kolberg.

4.11.2 An Extended Example

We create a space of 3-adic modular forms:

```
sage: M = OverconvergentModularForms(3, 8, 1/6, prec=60)
```

Creating an element directly as a linear combination of basis vectors.

```
sage: f1 = M.3 + M.5; f1.q_expansion()
27*q^3 + 1055916/1093*q^4 + 19913121/1093*q^5 + 268430112/1093*q^6 + ...
sage: f1.coordinates(8)
[0, 0, 0, 1, 0, 1, 0, 0]
```

We can coerce from elements of classical spaces of modular forms:

```
sage: f2 = M(CuspForms(3, 8).0); f2
3-adic overconvergent modular form of weight-character 8 with q-expansion q + 6*q^2 - ...
```

We can coerce from elements of classical spaces of modular forms:
We express this in a basis, and see that the coefficients go to zero very fast:

```python
sage: [x.valuation(3) for x in f2.coordinates(60)]
```

This form has more level at $p$, and hence is less overconvergent:

```python
sage: f3 = M(CuspForms(9, 8).0); [x.valuation(3) for x in f3.coordinates(60)]
[+Infinity, -1, -1, 0, -4, -4, -2, -3, 0, 0, -1, -1, 1, 0, 3, 3, 3, 3, 5, 3, 7, 6, 6,... -8, 7, 10, 10, 8, 8, 10, 9, 12, 12, 12, 14, 12, 14, 17, 16, 15, 15, 17, 16, 19, 19, 18,... -18, 20, 19, 22, 22, 22, 24, 24, 21, 25, 26, 24, 24]
```

An error will be raised for forms which are not sufficiently overconvergent:

```python
sage: M(CuspForms(27, 8).0)
Traceback (most recent call last):
... ValueError: Form is not overconvergent enough (form is only 1/12-overconvergent)
```

Let's compute some Hecke operators. Note that the coefficients of this matrix are $p$-adically tiny:

```python
sage: M.hecke_matrix(3, 4).change_ring(Qp(3,prec=1))
[ 1 + O(3) 0 0 0]
[ 0 2*3^3 + O(3^4) 2*3^3 + O(3^4) 3^2 + O(3^3)]
[ 0 2*3^7 + O(3^8) 2*3^8 + O(3^9) 3^6 + O(3^7)]
[ 0 2*3^10 + O(3^11) 2*3^10 + O(3^11) 2*3^9 + O(3^10)]
```

We compute the eigenfunctions of a 4x4 truncation:

```python
sage: efuncs = M.eigenfunctions(4)
sage: for i in [1..3]:
    ....:    print(efuncs[i].q_expansion(prec=4).change_ring(Qp(3,prec=20)))
(1 + O(3^20))*q + (2*3 + 3^15 + 3^16 + 3^17 + 2*3^19 + 2*3^20 + O(3^21))*q^2 + (2*3^3 + 2*3^4 + 2*3^5 + 2*3^6 + 2*3^7 + 2*3^8 + 2*3^9 + 2*3^10 + 2*3^11 + 2*3^12 + 2*3^13 + 2*3^14 + 2*3^15 + 2*3^16 + 3*17 + 2*3^19 + 3*21 + 3*22 + O(3^23))*q^3 + O(q^4)
(1 + O(3^20))*q + (3 + 2*3^2 + 3*3 + 3^4 + 3*12 + 3*13 + 2*3^14 + 3*15 + 2*3^17 + 3*18 + 3^19 + 3*20 + O(3^21))*q^2 + (3^7 + 3*13 + 2*3^14 + 2*3^15 + 3*16 + 3*17 + 2*3^18 + 3^20 + 2*3^21 + 2*3^22 + 2*3^23 + 2*3^25 + O(3^27))*q^3 + O(q^4)
(1 + O(3^20))*q + (2*3 + 3*3 + 2*3^4 + 3*6 + 2*3^8 + 3*9 + 3*10 + 2*3^11 + 2*3^13 + 3*16 + 3*18 + 3*19 + 3*20 + 0(3^21))*q^2 + (3^9 + 2*3^12 + 3*15 + 3*17 + 3*18 + 3*19 + 3*20 + 3*22 + 2*3^23 + 2*3^27 + 2*3^28 + O(3^29))*q^3 + O(q^4)
```

The first eigenfunction is a classical cusp form of level 3:

```python
sage: (efuncs[1] - M(CuspForms(3, 8).0)).valuation()
13
```

The second is an Eisenstein series!

```python
sage: (efuncs[2] - M(EisensteinForms(3, 8).1)).valuation()
10
```

4.11. Overconvergent $p$-adic modular forms for small primes
The third is a genuinely new thing (not a classical modular form at all); the coefficients are almost certainly not algebraic over \( \mathbb{Q} \). Note that the slope is 9, so Coleman’s classicality criterion (forms of slope \( < k - 1 \) are classical) does not apply.

```python
sage: a3 = efuncs[3].q_expansion()[3]; a3
3^9 + 2*3^12 + 3^15 + 3^17 + 3^18 + 3^19 + 3^20 + 2*3^22 + 2*3^23 + 2*3^27 + 2*3^28 + 3^...
```

```python
sage: efuncs[3].slope()
9
```

class `sage.modular.overconvergent.genus0.OverconvergentModularFormElement`

Bases: `ModuleElement`

A class representing an element of a space of overconvergent modular forms.

**EXAMPLES:**

```python
sage: K.<w> = Qp(5).extension(x^7 - 5); s = OverconvergentModularForms(5, 6, 1/21, base_ring=K).0
sage: s == loads(dumps(s))
True
```

**additive_order()**

Return the additive order of this element (required attribute for all elements deriving from `sage.modules.ModuleElement`).

**EXAMPLES:**

```python
sage: M = OverconvergentModularForms(13, 10, 1/2, base_ring = Qp(13).extension(x^2 - 13, names='a'))
```

```python
sage: M.gen(0).additive_order()
+Infinity
```

```python
sage: M(0).additive_order()
1
```

**base_extend(R)**

Return a copy of self but with coefficients in the given ring.

**EXAMPLES:**

```python
sage: M = OverconvergentModularForms(7, 10, 1/2, prec=5)
sage: f = M.1
sage: f.base_extend(Qp(7, 4))
```

```python
7-adic overconvergent modular form of weight-character 10 with q-expansion (7 +0(7^5))q + (6*7 + 4*7^2 + 7^3 + 6*7^4 + 0(7^5))q^2 + (5*7 + 5*7^2 + 7^4 +0(7^5))q^3 + (7^2 + 4*7^3 + 3*7^4 + 2*7^5 + 0(7^6))q^4 + 0(q^5)
```

**coordinates(prec=None)**

Return the coordinates of this modular form in terms of the basis of this space.

**EXAMPLES:**
sage: M = OverconvergentModularForms(3, 0, 1/2, prec=15)
sage: f = (M.0 + M.3); f.coordinates()
[1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
sage: f.coordinates(6)
[1, 0, 0, 1, 0, 0]
sage: OverconvergentModularForms(3, 0, 1/6)(f).coordinates(6)
[1, 0, 0, 729, 0, 0]
sage: f.coordinates(100)
Traceback (most recent call last):
  ... ValueError: Precision too large for space

```
eigenvalue()

Return the $U_p$-eigenvalue of this eigenform. Raises an error unless this element was explicitly flagged as
an eigenform, using the _notify_eigen function.

EXAMPLES:
```
sage: M = OverconvergentModularForms(3, 0, 1/2)
sage: f = M.eigenfunctions(3)[1]
sage: f.eigenvalue()
3^{-2} + 3^{-4} + 2*3^{-6} + 3^{-7} + 3^{-8} + 2*3^{-10} + 3^{-12} + 3^{-16} + 2*3^{-17} + 3^{-18} +

```
gexp()

Return the formal power series in $g$ corresponding to this overconvergent modular form (so the result is $F$
where this modular form is $E_k' \times F(g)$, where $g$ is the appropriately normalised parameter of $X_0(p)$).

EXAMPLES:
```
sage: M = OverconvergentModularForms(3, 0, 1/2)
sage: f = M.eigenfunctions(3)[1]
sage: f.gexp()
(3^-3 + O(3^95))*g + (3^-1 + 1 + 2*3 + 3^2 + 2*3^3 + 3^5 + 3^7 + 3^10 + 3^11 +

```
governing_term(r)

The degree of the series term with largest norm on the $r$-overconvergent region.

EXAMPLES:
```

4.11. Overconvergent $p$-adic modular forms for small primes
```python
sage: o = OverconvergentModularForms(3, 0, 1/2)
sage: f = o.eigenfunctions(10)[1]
sage: f.governing_term(1/2)
1
```

**is_eigenform()**

Return `True` if this is an eigenform. At present this returns `False` unless this element was explicitly flagged as an eigenform, using the `_notify_eigen` function.

**EXAMPLES:**

```python
sage: M = OverconvergentModularForms(3, 0, 1/2)
sage: f = M.eigenfunctions(3)[1]
sage: f.is_eigenform()
True
sage: M.gen(4).is_eigenform()
False
```

**is_integral()**

Test whether or not this element has $q$-expansion coefficients that are $p$-adically integral. This should always be the case with eigenfunctions, but sometimes if $n$ is very large this breaks down for unknown reasons!

**EXAMPLES:**

```python
sage: M = OverconvergentModularForms(2, 0, 1/3)
sage: q = QQ[['q']].gen()
sage: M(q - 17*q^2 + O(q^3)).is_integral()
True
sage: M(q - q^2/2 + 6*q^7 + O(q^9)).is_integral()
False
```

**prec()**

Return the series expansion precision of this overconvergent modular form. (This is not the same as the $p$-adic precision of the coefficients.)

**EXAMPLES:**

```python
sage: OverconvergentModularForms(5, 6, 1/3, prec=15).gen(1).prec()
15
```

**prime()**

If this is a $p$-adic modular form, return $p$.

**EXAMPLES:**

```python
sage: OverconvergentModularForms(2, 0, 1/2).an_element().prime()
2
```

**q_expansion(prec=None)**

Return the $q$-expansion of self, to as high precision as it is known.

**EXAMPLES:**

```python
sage: OverconvergentModularForms(3, 4, 1/2).gen(0).q_expansion()
1 - 120/13*q - 1080/13*q^2 - 120/13*q^3 - 8760/13*q^4 - 15120/13*q^5 - 1080/
```

(continues on next page)
r_ord\(r\)
The \(p\)-adic valuation of the norm of self on the \(r\)-overconvergent region.

EXAMPLES:

```
sage: o = OverconvergentModularForms(3, 0, 1/2)
sage: t = o([1, 1, 1/3])
sage: t.r_ord(1/2)
1
sage: t.r_ord(2/3)
3
```

slope()
Return the slope of this eigenform, i.e. the valuation of its \(U_p\)-eigenvalue. Raises an error unless this element was explicitly flagged as an eigenform, using the \_notify_eigen function.

EXAMPLES:

```
sage: M = OverconvergentModularForms(3, 0, 1/2)
sage: f = M.eigenfunctions(3)[1]
sage: f.slope()
2
sage: M.gen(4).slope()
Traceback (most recent call last):
...
TypeError: slope only defined for eigenfunctions
```

valuation()
Return the \(p\)-adic valuation of this form (i.e. the minimum of the \(p\)-adic valuations of its coordinates).

EXAMPLES:

```
sage: M = OverconvergentModularForms(3, 0, 1/2)
sage: (M.7).valuation()
0
sage: (3^18 * (M.2)).valuation()
18
```

valuation_plot\(r_{max}=None\)
Draw a graph depicting the growth of the norm of this overconvergent modular form as it approaches the boundary of the overconvergent region.

EXAMPLES:

```
sage: o = OverconvergentModularForms(3, 0, 1/2)
sage: f = o.eigenfunctions(4)[1]
sage: f.valuation_plot()
Graphics object consisting of 1 graphics primitive
weight()

Return the weight of this overconvergent modular form.

EXAMPLES:

```
sage: M = OverconvergentModularForms(13, 10, 1/2, base_ring = Qp(13).
˓→extension(x^2 - 13,names='a'))
sage: M.gen(0).weight()
10
```

sage.modular.overconvergent.genus0.OverconvergentModularForms

Create a space of overconvergent $p$-adic modular forms of level $\Gamma_0(p)$, over the given base ring. The base ring need not be a $p$-adic ring (the spaces we compute with typically have bases over $\mathbb{Q}$).

INPUT:

- prime - a prime number $p$, which must be one of the primes $\{2, 3, 5, 7, 13\}$, or the congruence subgroup $\Gamma_0(p)$ where $p$ is one of these primes.
- weight - an integer (which at present must be 0 or $\geq 2$), the weight.
- radius - a rational number in the interval $(0, \frac{p}{p+1})$, the radius of overconvergence.
- base_ring (default: $\mathbb{Q}$), a ring over which to compute. This need not be a $p$-adic ring.
- prec - an integer (default: 20), the number of $q$-expansion terms to compute.
- char - a Dirichlet character modulo $p$ or None (the default). Here None is interpreted as the trivial character modulo $p$.

The character $\chi$ and weight $k$ must satisfy $(-1)^k = \chi(-1)$, and the base ring must contain an element $v$ such that $\text{ord}_p(v) = \frac{12r}{p+1}$ where $r$ is the radius of overconvergence (and $\text{ord}_p$ is normalised so $\text{ord}_p(p) = 1$).

EXAMPLES:

```
sage: OverconvergentModularForms(3, 0, 1/2)
Space of 3-adic 1/2-overconvergent modular forms of weight-character 0 over Rational Field
sage: OverconvergentModularForms(3, 16, 1/2)
Space of 3-adic 1/2-overconvergent modular forms of weight-character 16 over Rational Field
sage: OverconvergentModularForms(3, 3, 1/2, char = DirichletGroup(3,QQ).0)
Space of 3-adic 1/2-overconvergent modular forms of weight-character (3, 3, [-1]) over Rational Field
```

class sage.modular.overconvergent.genus0.OverconvergentModularFormsSpace

A space of overconvergent modular forms of level $\Gamma_0(p)$, where $p$ is a prime such that $X_0(p)$ has genus 0.

Elements are represented as power series, with a formal power series $F$ corresponding to the modular form $E_k^* \times F(g)$ where $E_k^*$ is the $p$-deprived Eisenstein series of weight-character $k$, and $g$ is a uniformiser of $X_0(p)$ normalised so that the $r$-overconvergent region $X_0(p)_{\geq r}$ corresponds to $|g| \leq 1$. 

388 Chapter 4. Miscellaneous Modules (to be sorted)
**base_extend**(ring)

Return the base extension of self to the given base ring. There must be a canonical map to this ring from the current base ring, otherwise a TypeError will be raised.

EXAMPLES:

```python
sage: M = OverconvergentModularForms(2, 0, 1/2, base_ring = Qp(2))
sage: M.base_extend(Qp(2).extension(x^2 - 2, names="w"))
Space of 2-adic 1/2-overconvergent modular forms of weight-character 0 over 2-adic Eisenstein Extension ...
sage: M.base_extend(QQ)
Traceback (most recent call last):
...
TypeError: Base extension of self (over '2-adic Field with capped relative precision 20') to ring 'Rational Field' not defined.
```

**change_ring**(ring)

Return the space corresponding to self but over the given base ring.

EXAMPLES:

```python
sage: M = OverconvergentModularForms(2, 0, 1/2)
sage: M.change_ring(Qp(2))
Space of 2-adic 1/2-overconvergent modular forms of weight-character 0 over 2-adic Field with ...
```

**character()**

Return the character of self. For overconvergent forms, the weight and the character are unified into the concept of a weight-character, so this returns exactly the same thing as self.weight().

EXAMPLES:

```python
sage: OverconvergentModularForms(3, 0, 1/2).character()
0
sage: type(OverconvergentModularForms(3, 0, 1/2).character())
<class '...weightspace.AlgebraicWeight'>
sage: OverconvergentModularForms(3, 3, 1/2, char=DirichletGroup(3,QQ).0).character()
(3, 3, [-1])
```

**coordinate_vector**(x)

Write x as a vector with respect to the basis given by self.basis(). Here x must be an element of this space or something that can be converted into one. If x has precision less than the default precision of self, then the returned vector will be shorter.

EXAMPLES:

```python
sage: M = OverconvergentModularForms(Gamma0(3), 0, 1/3, prec=4)
sage: M.coordinate_vector(M.gen(2))
(0, 0, 1, 0)
sage: q = QQ[['q']].gen(); M.coordinate_vector(q - q^2 + O(q^4))
(0, 1/9, -13/81, 74/243)
sage: M.coordinate_vector(q - q^2 + O(q^3))
(0, 1/9, -13/81)
```
**cps_u(n, use_recurrence=False)**

Compute the characteristic power series of $U_p$ acting on self, using an $n \times n$ matrix.

**EXAMPLES:**

```python
sage: OverconvergentModularForms(3, 16, 1/2, base_ring=Qp(3)).cps_u(4)
1 + O(3^20) + (2 + 2*3 + 2*3^2 + 2*3^4 + 3^5 + 3^6 + 3^7 + 3^11 + 3^12 + 2*3^14...
→+ 3*16 + 3*18 + O(3^19))*T + (2*3^3 + 3^5 + 3^6 + 3^7 + 2*3^8 + 2*3^9 + 2*3^...
→- 10 + 3*11 + 3*12 + 2*3^13 + 2*3^16 + 2*3^18 + O(3^19))*T^2 + (2*3^15 + 2*3^16...
→- 2*3^19 + 2*3^20 + 2*3^21 + O(3^22))*T^3 + (3*17 + 2*3^18 + 3*19 + 3*20 + 3^...
→- 22 + 2*3^23 + 2*3^25 + 3*26 + O(3^27))*T^4
```

**eigenfunctions(n, F=None, exact_arith=True)**

Calculate approximations to eigenfunctions of self.

These are the eigenfunctions of self.hecke_matrix(p, n), which are approximations to the true eigenfunctions. Returns a list of OverconvergentModularFormElement objects, in increasing order of slope.

**INPUT:**

- **n** - integer. The size of the matrix to use.
- **F** - None, or a field over which to calculate eigenvalues. If the field is None, the current base ring is used. If the base ring is not a $p$-adic ring, an error will be raised.
- **exact_arith** - True or False (default True). If True, use exact rational arithmetic to calculate the matrix of the $U$ operator and its characteristic power series, even when the base ring is an inexact $p$-adic ring. This is typically slower, but more numerically stable.

**NOTE:** Try using set_verbose(1, 'sage/modular/overconvergent') to get more feedback on what is going on in this algorithm. For even more feedback, use 2 instead of 1.

**EXAMPLES:**

```python
sage: X = OverconvergentModularForms(2, 2, 1/6).eigenfunctions(8, Qp(2, 100))
sage: X[1]
2-adic overconvergent modular form of weight-character 2 with q-expansion (1 +...
→+ 2*4 + 2*5 + 2*9 + 2*10 + 2*12 + 2*13 + 2*15 + 2*17 + 2*19 + 2^...
→- 20 + 2*21 + 2*23 + 2*28 + 2*30 + 2*31 + 2*32 + 2*34 + 2*36 + 2*37 + 2*39 + 2^...
→- 40 + 2*43 + 2*44 + 2*45 + 2*47 + 2*48 + 2*52 + 2*53 + 2*54 + 2*55 + 2*56 + 2^...
→- 58 + 2*59 + 2*60 + 2*61 + 2*67 + 2*68 + 2*70 + 2*71 + 2*72 + 2*74 + 2*76 + ...
→- O(2^78))*T^4 + (2*2 + 2*7 + 2*8 + 2*9 + 2*12 + 2*13 + 2*16 + 2*17 + 2*21 + 2^...
→- 23 + 2*25 + 2*28 + 2*33 + 2*34 + 2*36 + 2*37 + 2*42 + 2*45 + 2*47 + 2*49 + 2^...
→- 50 + 2*51 + 2*54 + 2*55 + 2*58 + 2*60 + 2*61 + 2*67 + 2*71 + 2*72 + O(2^...
→- 76))*T^7 + (2*8 + 2*11 + 2*14 + 2*19 + 2*21 + 2*22 + 2*24 + 2*25 + 2*26 + 2^...
→- 27 + 2*28 + 2*29 + 2*32 + 2*33 + 2*35 + 2*36 + 2*44 + 2*45 + 2*46 + 2*47 + 2^...
```

(continues on next page)
\[ \begin{align*}
&49 + 2^50 + 2^53 + 2^54 + 2^55 + 2^57 + 2^60 + 2^63 + 2^66 + 2^67 + 2^69 + 2^74 + 2^76 + 2^79 + 2^80 + 2^81 + 0(2^82))q^4 + (2 + 2^2 + 2^9 + 2^13 + & \\
&+ 2^15 + 2^17 + 2^19 + 2^21 + 2^23 + 2^26 + 2^27 + 2^28 + 2^30 + 2^33 + 2^34 + & \\
&+ 2^35 + 2^36 + 2^37 + 2^38 + 2^39 + 2^41 + 2^42 + 2^43 + 2^45 + 2^58 + 2^59 + & \\
&+ 2^60 + 2^61 + 2^62 + 2^63 + 2^65 + 2^66 + 2^68 + 2^69 + 2^71 + 2^72 + 0(2^75))q^5 + (2^6 + 2^7 + 2^15 + 2^16 + 2^21 + 2^24 + 2^25 + 2^28 + 2^29 + 2^33 + & \\
&+ 2^34 + 2^37 + 2^44 + 2^45 + 2^48 + 2^50 + 2^51 + 2^54 + 2^55 + 2^57 + 2^58 + & \\
&+ 2^59 + 2^60 + 2^64 + 2^69 + 2^71 + 2^73 + 2^75 + 2^78 + 0(2^80))q^6 + (2^3. & \\
&+ 2^8 + 2^9 + 2^10 + 2^11 + 2^12 + 2^14 + 2^15 + 2^17 + 2^19 + 2^20 + 2^21 + & \\
&+ 2^23 + 2^25 + 2^26 + 2^34 + 2^37 + 2^38 + 2^39 + 2^40 + 2^41 + 2^45 + 2^47 + & \\
&+ 2^49 + 2^51 + 2^53 + 2^54 + 2^55 + 2^57 + 2^58 + 2^59 + 2^60 + 2^61 + 2^66 + & \\
&+ 2^69 + 2^70 + 2^71 + 2^74 + 2^76 + 0(2^77))q^7 + 0(q^8)
\end{align*} \]

\begin{Verbatim}
sage: [x.slope() for x in X]
[0, 4, 8, 14, 16, 18, 26, 30]
\end{Verbatim}

gen()

Return the \( i \)th module generator of \( \text{self} \).

**EXAMPLES:**

\begin{Verbatim}
sage: M = OverconvergentModularForms(3, 2, 1/2, prec=4)
sage: M.gen(0)
3-adic overconvergent modular form of weight-character 2 with q-expansion 1 + \\
12*q + 36*q^2 + 12*q^3 + O(q^4)
sage: M.gen(1)
3-adic overconvergent modular form of weight-character 2 with q-expansion 27*q + \\
648*q^2 + 7290*q^3 + O(q^4)
sage: M.gen(30)
3-adic overconvergent modular form of weight-character 2 with q-expansion 0(q^4)
\end{Verbatim}

gens()

Return a generator object that iterates over the (infinite) set of basis vectors of \( \text{self} \).

**EXAMPLES:**

\begin{Verbatim}
sage: o = OverconvergentModularForms(3, 12, 1/2)
sage: t = o.gens()
sage: next(t)
3-adic overconvergent modular form of weight-character 12 with q-expansion 1 - \\
32760/61203943*q - 67125240/61203943*q^2 - ... 
sage: next(t)
3-adic overconvergent modular form of weight-character 12 with q-expansion 27*q + \\
19829193012/61203943*q^2 + 146902585770/61203943*q^3 + ...
\end{Verbatim}

gens_dict()

Return a dictionary mapping the names of generators of this space to their values. (Required by parent class definition.) As this does not make any sense here, this raises a TypeError.

**EXAMPLES:**

\begin{Verbatim}
sage: M = OverconvergentModularForms(2, 4, 1/6)
sage: M.gens_dict()
Traceback (most recent call last):
(continues on next page)
\end{Verbatim}

4.11. Overconvergent \( p \)-adic modular forms for small primes
Modular Forms, Release 10.0

(continued from previous page)

```
... 
TypeError: gens_dict does not make sense as number of generators is infinite
```

**hecke_matrix\((m, n, use\_recurrence=False, exact\_arith=False)\)**

Calculate the matrix of the \(T_m\) operator in the basis of this space, truncated to an \(n \times n\) matrix. Conventions are that operators act on the left on column vectors (this is the opposite of the conventions of the sage.modules.matrix_morphism class!) Uses naive \(q\)-expansion arguments if use\_recurrence=False and uses the Kolberg style recurrences if use\_recurrence=True.

The argument “exact\_arith” causes the computation to be done with rational arithmetic, even if the base ring is an inexact \(p\)-adic ring. This is useful as there can be precision loss issues (particularly with use\_recurrence=False).

**EXAMPLES:**

```
sage: OverconvergentModularForms(2, 0, 1/2).hecke_matrix(2, 4)
[ 1 0 0 0]
[ 0 24 64 0]
[ 0 32 1152 4608]
[ 0 0 3072 61440]
sage: OverconvergentModularForms(2, 12, 1/2, base_ring=pAdicField(2)).hecke_matrix(2, 3) * (1 + O(2^2))
[ 1 + O(2^2) 0 0]
[ 0 2^3 + O(2^5) 2^6 + O(2^8)]
[ 0 2^4 + O(2^6) 2^7 + 2^8 + O(2^9)]
sage: OverconvergentModularForms(2, 12, 1/2, base_ring=pAdicField(2)).hecke_matrix(2, 3, exact\_arith=True)
[ 1 0 ␣→ 0]
[ 0 33881928/1414477 ␣→ 64]
[ 0 192898739923312/2000745183529/1414477]
```

**hecke_operator\((f, m)\)**

Given an element \(f\) and an integer \(m\), calculates the Hecke operator \(T_m\) acting on \(f\).

The input may be either a “bare” power series, or an OverconvergentModularFormElement object; the return value will be of the same type.

**EXAMPLES:**

```
sage: M = OverconvergentModularForms(3, 0, 1/2)
sage: f = M.1
sage: M.hecke_operator(f, 3)
3-adic overconvergent modular form of weight-character 0 with q-expansion
2430*q + 265356*q^2 + 10670373*q^3 + 249948828*q^4 + 4113612864*q^5 +...
-> 52494114852*q^6 + O(q^7)
sage: M.hecke_operator(f.q_expansion(), 3)
2430*q + 265356*q^2 + 10670373*q^3 + 249948828*q^4 + 4113612864*q^5 +...
-> 52494114852*q^6 + O(q^7)
```

**is_exact()**

True if elements of this space are represented exactly, i.e., there is no precision loss when doing arithmetic. As this is never true for overconvergent modular forms spaces, this returns False.
EXAMPLES:

```sage
sage: OverconvergentModularForms(13, 12, 0).is_exact()
False
```

`ngens()`

The number of generators of self (as a module over its base ring), i.e. infinity.

EXAMPLES:

```sage
sage: M = OverconvergentModularForms(2, 4, 1/6)
sage: M.ngens()
+Infinity
```

`normalising_factor()`

The normalising factor $c$ such that $g = cf$ is a parameter for the $r$-overconvergent disc in $X_0(p)$, where $f$ is the standard uniformiser.

EXAMPLES:

```sage
sage: L.<w> = Qp(7).extension(x^2 - 7)
sage: OverconvergentModularForms(7, 0, 1/4, base_ring=L).normalising_factor()
w + O(w^41)
```

`prec()`

Return the series precision of self. Note that this is different from the $p$-adic precision of the base ring.

EXAMPLES:

```sage
sage: OverconvergentModularForms(3, 0, 1/2).prec()
20
sage: OverconvergentModularForms(3, 0, 1/2, prec=40).prec()
40
```

`prime()`

Return the residue characteristic of self, i.e. the prime $p$ such that this is a $p$-adic space.

EXAMPLES:

```sage
sage: OverconvergentModularForms(5, 12, 1/3).prime()
5
```

`radius()`

The radius of overconvergence of this space.

EXAMPLES:

```sage
sage: OverconvergentModularForms(3, 0, 1/3).radius()
1/3
```

`recurrence_matrix(use_smithline=True)`

Return the recurrence matrix satisfied by the coefficients of $U$, that is a matrix $R = (r_{rs})_{r,s=1...p}$ such that $u_{ij} = \sum_{r,s=1}^{p} r_{rs} u_{i-r,j-s}$. Uses an elegant construction which I believe is due to Smithline. See [Loe2007].

EXAMPLES:
```python
sage: OverconvergentModularForms(2, 0, 0).recurrence_matrix()
[ 48  1]
[4096 0]
sage: OverconvergentModularForms(2, 0, 1/2).recurrence_matrix()
[ 48  64]
[ 64  0]
sage: OverconvergentModularForms(3, 0, 0).recurrence_matrix()
[ 270  36  1]
[26244  729  0]
[531441  0  0]
sage: OverconvergentModularForms(5, 0, 0).recurrence_matrix()
[ 1575  1300  315  30  1]
[162500  39375  3750  125  0]
[4921875  468750  15625  0  0]
[58593750  1953125  0  0  0]
[244140625  0  0  0  0]
sage: OverconvergentModularForms(7, 0, 0).recurrence_matrix()
[ 4018  8624  5915  1904  322  28  \rightarrow 1]
[ 422576  289835  93296  15778  1372  49  \rightarrow 0]
[ 14201915  4571504  773122  67228  2401  0  \rightarrow 0]
[ 224003696  37882978  3294172  117649  0  0  \rightarrow 0]
[ 1856265922  161414428  5764801  0  0  0  \rightarrow 0]
[ 7909306972  282475249  0  0  0  0  \rightarrow 0]
[13841287201  0  0  0  0  0  \rightarrow 0]
sage: OverconvergentModularForms(13, 0, 0).recurrence_matrix()
[ 15145  124852  354536 ...]
```

**slopes**\(n, use\_recurrence=False\)
Compute the slopes of the \(U_p\) operator acting on self, using an \(n\times n\) matrix.

**EXAMPLES:**

```python
sage: OverconvergentModularForms(5,2,1/3,base\_ring=Qp(5),prec=100).slopes(5)
[0, 2, 5, 6, 9]
sage: OverconvergentModularForms(2,1,1/3,char=DirichletGroup(4,QQ).0).slopes(5)
[0, 2, 4, 6, 8]
```

**weight()**
Return the character of self. For overconvergent forms, the weight and the character are unified into the concept of a weight-character, so this returns exactly the same thing as self.character().

**EXAMPLES:**

```python
sage: OverconvergentModularForms(3, 0, 1/2).weight()
0
sage: type(OverconvergentModularForms(3, 0, 1/2).weight())
<class '...weightspace.AlgebraicWeight'>
```
sage: OverconvergentModularForms(3, 3, 1/2, char=DirichletGroup(3,QQ).0).
˓→weight()
(3, 3, [-1])

zero()
Return the zero of this space.

EXAMPLES:

sage: K.<w> = Qp(13).extension(x^2-13); M = OverconvergentModularForms(13, 20,
˓→radius=1/2, base_ring=K)
sage: K.zero()
0

4.12 Atkin/Hecke series for overconvergent modular forms

This file contains a function `hecke_series()` to compute the characteristic series $P(t)$ modulo $p^m$ of the Atkin/Hecke operator $U_p$ upon the space of $p$-adic overconvergent modular forms of level $\Gamma_0(N)$. The input weight $k$ can also be a list $klist$ of weights which must all be congruent modulo $(p - 1)$.

Two optional parameters `modformsring` and `weightbound` can be specified, and in most cases for levels $N > 1$ they can be used to obtain the output more quickly. When $m \leq k - 1$ the output $P(t)$ is also equal modulo $p^m$ to the reverse characteristic polynomial of the Atkin operator $U_p$ on the space of classical modular forms of weight $k$ and level $\Gamma_0(Np)$. In addition, provided $m \leq (k - 2)/2$ the output $P(t)$ is equal modulo $p^m$ to the reverse characteristic polynomial of the Hecke operator $T_p$ on the space of classical modular forms of weight $k$ and level $\Gamma_0(N)$. The function is based upon the main algorithm in [Lau2011], and has linear running time in the logarithm of the weight $k$.

AUTHORS:

• Alan G.B. Lauder (2011-11-10): original implementation.
• David Loeffler (2011-12): minor optimizations in review stage.

EXAMPLES:

The characteristic series of the $U_{11}$ operator modulo 11$^{10}$ on the space of 11-adic overconvergent modular forms of level 1 and weight 10000:

sage: hecke_series(11, 1, 10000, 10)
10009319650*x^4 + 25618839103*x^3 + 6126165716*x^2 + 10120524732*x + 1

The characteristic series of the $U_5$ operator modulo 5$^5$ on the space of 5-adic overconvergent modular forms of level 3 and weight 1000:

sage: hecke_series(5, 3, 1000, 5)
1875*x^6 + 1250*x^5 + 1200*x^4 + 1385*x^3 + 1131*x^2 + 2533*x + 1

The characteristic series of the $U_7$ operator modulo 7$^5$ on the space of 7-adic overconvergent modular forms of level 5 and weight 1000. Here the optional parameter `modformsring` is set to True:

sage: hecke_series(7, 5, 1000, 5, modformsring=True) # long time (21s on sage.math,˓→2012)
12005*x^7 + 10633*x^6 + 6321*x^5 + 6216*x^4 + 5412*x^3 + 4927*x^2 + 4906*x + 1

4.12. Atkin/Hecke series for overconvergent modular forms 395
The characteristic series of the $U_{13}$ operator modulo $13^5$ on the space of 13-adic overconvergent modular forms of level 2 and weight 10000. Here the optional parameter weightbound is set to 4:

\[
\text{sage: } \text{hecke_series}(13, 2, 10000, 5, \text{weightbound}=4) \quad \# \text{ long time (17s on sage.math, 2012)}
\]

\[
325156x^5 + 109681x^4 + 188617x^3 + 220858x^2 + 269566x + 1
\]

A list containing the characteristic series of the $U_{23}$ operator modulo $23^{10}$ on the spaces of 23-adic overconvergent modular forms of level 1 and weights 1000 and 1022, respectively.

\[
\text{sage: } \text{hecke_series}(23, 1, [1000, 1022], 10)
\]

\[
[7204616645852x^6 + 2117949463923x^5 + 24152587827773x^4 + 31270783576528x^3 + \ldots + 3033636679797x^2 + 2919723547079x + 1, 32737396672905x^4 + 36141830902187x^3 + 16514246534976x^2 + \ldots + 3888609530878x + 1]
\]

\[
\text{sage.modular.overconvergent.hecke_series.complementary_spaces(N, p, k0, n, mdash, elldashp, elldash, modformsring, bound)}
\]

Returns a list $W$, each element in which is a list $W_i$ of $q$-expansions modulo $(p, q^{elldashp})$. The list $W_i$ is a basis for a choice of complementary space in level $\Gamma_0(N)$ and weight $k$ to the image of weight $k - (p - 1)$ forms under multiplication by the Eisenstein series $E_{p-1}$. The lists $W_i$ play the same role as $W_i$ in Step 2 of Algorithm 2 in [Lau2011]. (The parameters $k0$, $n$, $mdash$, $elldashp = elldash * p$ are defined as in Step 1 of that algorithm when this function is used in \text{hecke_series}().) However, the complementary spaces are computed in a different manner, combining a suggestion of David Loeffler with one of John Voight. That is, one builds these spaces recursively using random products of forms in low weight, first searching for suitable products modulo $(p, q^{elldash})$, and then later reconstructing only the required products to the full precision modulo $(p, q^{elldashp})$. The forms in low weight are chosen from either bases of all forms up to weight bound or from a (tentative) generating set for the ring of all modular forms, according to whether \text{modformsring} is False or True.

INPUT:

- $N$ – positive integer at least 2 and not divisible by $p$ (level).
- $p$ – prime at least 5.
- $k0$ – integer in range 0 to $p - 1$.
- $n$, $mdash$, $elldashp$, $elldash$ – positive integers.
- \text{modformsring} – True or False.
- \text{bound} – positive (even) integer (ignored if \text{modformsring} is True).

OUTPUT:

- list of lists of $q$-expansions modulo $(p^{mdash}, q^{elldashp})$.

EXAMPLES:

\[
\text{sage: from sage.modular.overconvergent.hecke_series import complementary_spaces}
\]

\[
\text{sage: complementary_spaces(2, 5, 0, 3, 2, 5, True, 6)} \quad \# \text{ random}
\]

\[
[[1], [1 + 23q + 24q^2 + 19q^3 + 7q^4 + O(q^5)], [1 + 21q + 2q^2 + 17q^3 + 14q^4 + O(q^5)], [1 + 19q + 9q^2 + 11q^3 + 9q^4 + O(q^5)]]
\]

\[
\text{sage: complementary_spaces(2, 5, 0, 3, 2, 5, False, 6)} \quad \# \text{ random}
\]

\[
[[1], [3 + 4q + 2q^2 + 12q^3 + 11q^4 + O(q^5)]]
\]
sage.modular.overconvergent.hecke_series.complementary_spaces_modp(N, p, k0, n, elldash, LWBModp, bound)

Returns a list of lists of lists \([j, a]\). The pairs \([j, a]\) encode the choice of the \(a\)-th element in the \(j\)-th list of the input LWBModp, i.e., the \(a\)-th element in a particular basis modulo \((p, q^{\text{elldash}})\) for the space of modular forms of level \(\Gamma_0(N)\) and weight \(2(j+1)\). The list \([[j_1, a_1], \ldots, [j_r, a_r]]\) then encodes the product of the \(r\) modular forms associated to each \([j_i, a_i]\); this has weight \(k + (p - 1)i\) for some \(0 \leq i \leq n\); here the \(i\) is such that this list of lists occurs in the \(i\)th list of the output. The \(i\)th list of the output thus encodes a choice of basis for the complementary space \(W_i\) which occurs in Step 2 of Algorithm 2 in [Lau2011]. The idea is that one searches for this space \(W_i\) first modulo \((p, q^{\text{elldash}})\) and then, having found the correct products of generating forms, one can reconstruct these spaces modulo \((p^{\text{elldash}}, q^{\text{elldash}p})\) using the output of this function. (This idea is based upon a suggestion of John Voight.)

**INPUT:**
- \(N\) – positive integer at least 2 and not divisible by \(p\) (level).
- \(p\) – prime at least 5.
- \(k0\) – integer in range 0 to \(p - 1\).
- \(n, \text{elldash}\) – positive integers.
- \(LWBModp\) – list of lists of \(q\)-expansions over \(GF(p)\).
- \(bound\) – positive even integer (twice the length of the list LWBModp).

**OUTPUT:**
- list of list of list of lists.

**EXAMPLES:**

```python
sage: from sage.modular.overconvergent.hecke_series import random_low_weight_bases, \n    \n    sage: LWB = random_low_weight_bases(2, 5, 2, 4, 6)
    sage: LWBModp = [[f.change_ring(Zmod(5)) for f in x] for x in LWB]
    sage: complementary_spaces_modp(2, 5, 0, 3, 4, LWBModp, 6) \# random, indirect...
    [[[[[]], [[[0, 0], [0, 0]]], [[[0, 0], [2, 1]]], [[[0, 0], [0, 0], [0, 0]], [2, 1]]]]
```

sage.modular.overconvergent.hecke_series.compute_G(p, F)

Given a power series \(F \in R[[q]]\), for some ring \(R\), and an integer \(p\), compute the quotient

\[
\frac{F(q)}{F(q^p)}.
\]

Used by \texttt{level1_UpGj()} and by \texttt{higher_level_UpGj()}, with \(F\) equal to the Eisenstein series \(E_{p-1}\).

**INPUT:**
- \(p\) – integer
- \(F\) – power series (with invertible constant term)

**OUTPUT:**
the power series \(F(q)/F(q^p)\), to the same precision as \(F\)

**EXAMPLES:**

4.12. Atkin/Hecke series for overconvergent modular forms
This function computes a list $W_i$ of q-expansions, together with an auxiliary quantity $h_j$ (see below) which is to be used on the next call of this function. (The precision is that of input q-expansions.)

The list $W_i$ is a certain subset of a basis of the modular forms of weight $k$ and level 1. Suppose $(a, b)$ is the pair of non-negative integers with $4a + 6b = k$ and $a$ minimal among such pairs. Then this space has a basis given by

$$\{ \Delta^j E_6^{b-2j} E_4^a : 0 \leq j < d \}$$

where $d$ is the dimension.

What this function returns is the subset of the above basis corresponding to $e \leq j < d$ where $e$ is the dimension of the space of modular forms of weight $k - (p - 1)$. This set is a basis for the complement of the image of the weight $k - (p - 1)$ forms under multiplication by $E_p$.

This function is used repeatedly in the construction of the Katz expansion basis. Hence considerable care is taken to reuse steps in the computation wherever possible: we keep track of powers of the form $h_j = \Delta / E_6^2$.

**INPUT:**

- $k$ – non-negative integer.
- $p$ – prime at least 5.
- $h$ – q-expansion of $h$ (to some finite precision).
- $h_j$ – q-expansion of $h_j$ where $j$ is the dimension of the space of modular forms of level 1 and weight $k - (p - 1)$ (to same finite precision).
- $E_4$ – q-expansion of $E_4$ (to same finite precision).
- $E_6$ – q-expansion of $E_6$ (to same finite precision).

The Eisenstein series q-expansions should be normalized to have constant term 1.

**OUTPUT:**

- list of q-expansions (to same finite precision), and q-expansion.

**EXAMPLES:**

```python
sage: from sage.modular.overconvergent.hecke_series import compute_Wi
sage: p = 17
sage: prec = 10
sage: k = 24
sage: S = Zmod(17^3)
sage: E4 = eisenstein_series_qexp(4, prec, K=S, normalization="constant")
```

(continues on next page)
\[ -1752q^9 + O(q^{10}), q^2 + 4865q^3 + 1080q^4 + 4612q^5 + 1343q^6 + 1689q^7 + \omega_1 \]
\[ -3876q^8 + 1381q^9 + O(q^{10}) \]
\[ q^3 + 2952q^4 + 1278q^5 + 3225q^6 + 1286q^7 + 589q^8 + 122q^9 + O(q^{10}) \]

```
sage: c == ([delta_qexp(10) * E6^2, delta_qexp(10)^2], h**3)
True
```

Footnotes:

1. Sage: Modular Forms, Release 10.0

### sage.modular.overconvergent.hecke_series.compute_elldash(p, N, k0, n)

Returns the “Sturm bound” for the space of modular forms of level \( \Gamma_0(N) \) and weight \( k_0 + n(p - 1) \).

See also:

- `sturm_bound()`

**INPUT:**

- \( p \) – prime.
- \( N \) – positive integer (level).
- \( k_0, n \) - non-negative integers not both zero.

**OUTPUT:**

- positive integer.

**EXAMPLES:**

```
sage: from sage.modular.overconvergent.hecke_series import compute_elldash
sage: compute_elldash(11, 5, 4, 10)
53
```

### sage.modular.overconvergent.hecke_series.ech_form(A, p)

Return echelon form of matrix \( A \) over the ring of integers modulo \( p^m \), for some prime \( p \) and \( m \geq 1 \).

**Todo:** This should be moved to `sage.matrix.matrix_modn_dense` at some point.

**INPUT:**

- \( A \) – matrix over Zmod(p^m) for some m
- \( p \) – prime p

**OUTPUT:** matrix over Zmod(p^m)

**EXAMPLES:**

```
sage: from sage.modular.overconvergent.hecke_series import ech_form
sage: A = MatrixSpace(Zmod(5 ** 3), 3)([1, 2, 3, 4, 5, 6, 7, 8, 9])
sage: ech_form(A, 5)
[1 2 3]
[0 1 2]
[0 0 0]
```

### sage.modular.overconvergent.hecke_series.hecke_series(p, N, klist, m, modformsring=False, weightbound=6)

Returns the characteristic series modulo \( p^m \) of the Atkin operator \( U_p \) acting upon the space of \( p \)-adic overconvergent modular forms of level \( \Gamma_0(N) \) and weight \( klist \).
The input \texttt{klist} may also be a list of weights congruent modulo \((p - 1)\), in which case the output is the corresponding list of characteristic series for each \(k\) in \texttt{klist}; this is faster than performing the computation separately for each \(k\), since intermediate steps in the computation may be reused.

If \texttt{modformsring} is True, then for \(N > 1\) the algorithm computes at one step \texttt{ModularFormsRing(N).generators()}. This will often be faster but the algorithm will default to \texttt{modformsring=False} if the generators found are not p-adically integral. Note that \texttt{modformsring} is ignored for \(N = 1\) and the ring structure of modular forms is always used in this case.

When \texttt{modformsring} is False and \(N > 1\), \texttt{weightbound} is a bound set on the weight of generators for a certain subspace of modular forms. The algorithm will often be faster if \texttt{weightbound=4}, but it may fail to terminate for certain exceptional small values of \(N\), when this bound is too small.

The algorithm is based upon that described in [Lau2011].

**INPUT:**

- \(p\) – a prime greater than or equal to 5.
- \(N\) – a positive integer not divisible by \(p\).
- \texttt{klist} – either a list of integers congruent modulo \((p - 1)\), or a single integer.
- \(m\) – a positive integer.
- \texttt{modformsring} – True or False (optional, default False). Ignored if \(N = 1\).
- \texttt{weightbound} – a positive even integer (optional, default 6). Ignored if \(N = 1\) or \texttt{modformsring} is True.

**OUTPUT:**

Either a list of polynomials or a single polynomial over the integers modulo \(p^m\).

**EXAMPLES:**

```sage
sage: hecke_series(5, 7, 10000, 5, modformsring=True) # long time (3.4s)
250*x^6 + 1825*x^5 + 2500*x^4 + 2184*x^3 + 1458*x^2 + 1157*x + 1
sage: hecke_series(7, 3, 10000, 3, weightbound=4)
196*x^4 + 294*x^3 + 197*x^2 + 341*x + 1
sage: hecke_series(19, 1, [10000, 10018], 5)
[1694173*x^4 + 2442526*x^3 + 1367943*x^2 + 1923654*x + 1,
130321*x^4 + 958816*x^3 + 2278233*x^2 + 1584827*x + 1]
```

Check that silly weights are handled correctly:

```sage
sage: hecke_series(5, 7, [2, 3], 5)
Traceback (most recent call last):
  ... ValueError: List of weights must be all congruent modulo p-1 = 4, but given list contains 2 and 3 which are not congruent
sage: hecke_series(5, 7, [3], 5)
[1]
```

\texttt{sage.modular.overconvergent.hecke_series.hecke_series_degree_bound}(\(p, N, k, m\))

Returns the Wan bound on the degree of the characteristic series of the Atkin operator on \(p\)-adic overconvergent modular forms of level \(\Gamma_0(N)\) and weight \(k\) when reduced modulo \(p^m\).

This bound depends only upon \(p, k \pmod{p - 1}\), and \(N\). It uses Lemma 3.1 in [Wan1998].

**INPUT:**
• $p$ – prime at least 5.
• $N$ – positive integer not divisible by $p$.
• $k$ – even integer.
• $m$ – positive integer.

OUTPUT:
A non-negative integer.

EXAMPLES:

```
sage: from sage.modular.overconvergent.hecke_series import hecke_series_degree_bound
sage: hecke_series_degree_bound(13,11,100,5)
39
```

```
sage: higher_level_UpGj(p, N, klist, m, modformsring, bound, extra_data=False)
```

Return a list $[A_k]$ of square matrices over $\text{IntegerRing}(p^m)$ parameterised by the weights $k$ in $\text{klist}$.

The matrix $A_k$ is the finite square matrix which occurs on input $p, k, N$ and $m$ in Step 6 of Algorithm 2 in [Lau2011].

Notational change from paper: In Step 1 following Wan we defined $j$ by $k = k_0 + j(p-1)$ with $0 \leq k_0 < p-1$. Here we replace $j$ by $\text{kdiv}$ so that we may use $j$ as a column index for matrices.)

INPUT:
• $p$ – prime at least 5.
• $N$ – integer at least 2 and not divisible by $p$ (level).
• $\text{klist}$ – list of integers all congruent modulo $(p-1)$ (the weights).
• $m$ – positive integer.
• $\text{modformsring}$ – True or False.
• $\text{bound}$ – (even) positive integer.
• $\text{extra_data}$ – (default: False) boolean.

OUTPUT:
• list of square matrices. If $\text{extra_data}$ is True, return also extra intermediate data, namely the matrix $E$ in [Lau2011] and the integers $\elldash$ and $mdash$.

EXAMPLES:

```
sage: from sage.modular.overconvergent.hecke_series import higher_level_UpGj
sage: A = Matrix([...])
sage: B = Matrix([...])
```
sage: C = higher_level_UpGj(5, 3, [4], 2, True, 6)
sage: len(C)
1
sage: C[0] in (A, B)
True
sage: len(higher_level_UpGj(5, 3, [4], 2, True, 6, extra_data=True))
4

sage.modular.overconvergent.hecke_series.higher_level_katz_exp(p, N, k0, m, dash, ellashp, modformsring, bound)

Returns a matrix $e$ of size $ell \times ellashp$ over the integers modulo $p^{dash}$ and the Eisenstein series $E_{p-1} = 1 + \ldots \mod (p^{dash}, q^{ellashp})$. The matrix $e$ contains the coefficients of the elements $e_{i,s}$ in the Katz expansions basis in Step 3 of Algorithm 2 in [Lau2011] when one takes as input to that algorithm $p, N, m$ and $k$ and define $k0, dash, n, ellash, ellashp = ell * dash$ as in Step 1.

INPUT:

- $p$ – prime at least 5.
- $N$ – positive integer at least 2 and not divisible by $p$ (level).
- $k0$ – integer in range 0 to $p - 1$.
- $m, dash, ellash, ellashp$ – positive integers.
- modformsring – True or False.
- bound – positive (even) integer.

OUTPUT:

- matrix and $q$-expansion.

EXAMPLES:

sage: from sage.modular.overconvergent.hecke_series import higher_level_katz_exp
sage: e, Ep1 = higher_level_katz_exp(5, 2, 0, 1, 2, 4, 20, True, 6)
sage: e
[0 1 18 23 19 19 6 9 17 7 3 17 12 8 22 8 11 19 1 5]
[0 0 1 11 20 16 0 4 18 15 24 6 15 23 5 18 7 15]
[0 0 0 1 4 16 23 13 6 5 23 5 2 16 4 18 10 23 5 15]
sage: Ep1
1 + 15*q + 10*q^2 + 20*q^3 + 20*q^4 + 15*q^5 + 5*q^6 + 10*q^7 +
5*q^9 + 10*q^10 + 5*q^11 + 10*q^12 + 20*q^13 + 15*q^14 + 20*q^15 + 15*q^16 +
10*q^17 + 20*q^18 + O(q^20)

sage.modular.overconvergent.hecke_series.is_valid_weight_list(klist, p)

This function checks that klist is a nonempty list of integers all of which are congruent modulo $(p - 1)$. Otherwise, it will raise a ValueError.

INPUT:

- klist – list of integers.
- p – prime.
EXAMPLES:

```python
sage: from sage.modular.overconvergent.hecke_series import is_valid_weight_list
sage: is_valid_weight_list([10, 20, 30], 11)
Traceback (most recent call last):
  ... ValueError: List of weights must be non-empty
sage: is_valid_weight_list([-3, 1], 5)
Traceback (most recent call last):
  ... ValueError: List of weights must be all congruent modulo p-1 = 4, but given list contains -3 and 2 which are not congruent
```

```
sage.modular.overconvergent.hecke_series.katz_expansions(k0, p, ellp, mdash, n)

Returns a list $e$ of $q$-expansions, and the Eisenstein series $E_{p-1} = 1 + \ldots$, all modulo $(p^{mdash}, q^{ellp})$. The list $e$ contains the elements $e_{i,s}$ in the Katz expansions basis in Step 3 of Algorithm 1 in [Lau2011] when one takes as input to that algorithm $p, m$ and $k$ and define $k0, mdash, n, ellp = ell * p$ as in Step 1.

INPUT:
- $k0$ – integer in range 0 to $p - 1$.
- $p$ – prime at least 5.
- $ellp, mdash, n$ – positive integers.

OUTPUT:
- list of $q$-expansions and the Eisenstein series $E_{p-1}$ modulo $(p^{mdash}, q^{ellp})$.

EXAMPLES:

```python
sage: from sage.modular.overconvergent.hecke_series import katz_expansions
sage: katz_expansions(0, 5, 10, 3, 4)
([1 + O(q^10), q + 6*q^2 + 27*q^3 + 98*q^4 + 37*q^5 + 81*q^7 + 85*q^8 + 62*q^9 + O(q^10)],
1 + 115*q + 35*q^2 + 95*q^3 + 20*q^4 + 115*q^5 + 105*q^6 + 60*q^7 + 25*q^8 + 55*q^9 + O(q^10))
```

```
sage.modular.overconvergent.hecke_series.level1_UpGj(p, klist, m, extra_data=False)

Return a list $[A_k]$ of square matrices over $\text{IntegerRing}(p^m)$ parameterised by the weights $k$ in klist.

The matrix $A_k$ is the finite square matrix which occurs on input $p, k$ and $m$ in Step 6 of Algorithm 1 in [Lau2011].

Notational change from paper: In Step 1 following Wan we defined $j$ by $k = k_0 + j(p - 1)$ with $0 \leq k_0 < p - 1$. Here we replace $j$ by $kdiv$ so that we may use $j$ as a column index for matrices.

INPUT:
- $p$ – prime at least 5.
- $klist$ – list of integers congruent modulo $(p - 1)$ (the weights).
- $m$ – positive integer.
- $extra_data$ – (default: False) boolean

OUTPUT:
• list of square matrices. If extra_data is True, return also extra intermediate data, namely the matrix $E$ in [Lau2011] and the integers $\text{elldash}$ and $\text{mdash}$.

EXAMPLES:

```
sage: from sage.modular.overconvergent.hecke_series import level1_UpGj
sage: level1_UpGj(7, [100], 5)
[[1 980 4802 0 0]
 [0 13727 14406 0 0]
 [0 13440 7203 0 0]
 [0 1995 4802 0 0]
 [0 9212 14406 0 0]]
sage: len(level1_UpGj(7, [100], 5, extra_data=True))
4
```

```
sage.modular.overconvergent.hecke_series.low_weight_bases(N, p, m, NN, weightbound)
Return a list of integral bases of modular forms of level $N$ and (even) weight at most weightbound, as $q$-
expansions modulo $(p^m, q^{NN})$.
These forms are obtained by reduction mod $p^m$ from an integral basis in Hermite normal form (so they are not
necessarily in reduced row echelon form mod $p^m$, but they are not far off).

INPUT:
• $N$ – positive integer (level).
• $p$ – prime.
• $m, NN$ – positive integers.
• weightbound – (even) positive integer.

OUTPUT:
• list of lists of $q$-expansions modulo $(p^m, q^{NN})$.

EXAMPLES:

```
sage: from sage.modular.overconvergent.hecke_series import low_weight_bases
sage: low_weight_bases(2, 5, 3, 5, 6)
[[1 + 24*q + 24*q^2 + 96*q^3 + 24*q^4 + O(q^5)],
 [1 + 115*q^2 + 35*q^4 + O(q^5), q + 8*q^2 + 28*q^3 + 64*q^4 + O(q^5)],
 [1 + 121*q^2 + 118*q^4 + O(q^5), q + 32*q^2 + 119*q^3 + 24*q^4 + O(q^5)]]
```

```
sage.modular.overconvergent.hecke_series.low_weight_generators(N, p, m, NN)
Returns a list of lists of modular forms, and an even natural number.
The first output is a list of lists of modular forms reduced modulo $(p^m, q^{NN})$ which generate the $(\mathbb{Z}/p^m\mathbb{Z})$
-algebra of mod $p^m$ modular forms of weight at most 8, and the second output is the largest weight among the
forms in the generating set.

We (Alan Lauder and David Loeffler, the author and reviewer of this patch) conjecture that forms of weight at
most 8 are always sufficient to generate the algebra of mod $p^m$ modular forms of all weights. (We believe 6 to be
sufficient, and we can prove that 4 is sufficient when there are no elliptic points, but using weights up to 8 acts
as a consistency check.)

INPUT:
• $N$ – positive integer (level).
• $p$ – prime.
• $m, NN$ – positive integers.

OUTPUT:

a tuple consisting of:

• a list of lists of $q$-expansions modulo $(p^m, q^{NN})$,
• an even natural number (twice the length of the list).

EXAMPLES:

```
sage: from sage.modular.overconvergent.hecke_series import low_weight_generators
sage: low_weight_generators(3, 7, 3, 10)
([(1 + 12*q + 36*q^2 + 12*q^3 + 84*q^4 + 72*q^5 + 36*q^6 + 96*q^7 + 180*q^8 + 12*q^9 + O(q^10)),
  [1 + 240*q^3 + 102*q^6 + 203*q^9 + O(q^10)],
  [1 + 182*q^3 + 175*q^6 + 161*q^9 + O(q^10))], 6)
```

```
sage: low_weight_generators(11, 5, 3, 10)
([(1 + 12*q^2 + 12*q^3 + 12*q^4 + 12*q^5 + 24*q^6 + 24*q^7 + 36*q^8 + 36*q^9 + O(q^10)),
  [q + 123*q^2 + 124*q^3 + 2*q^4 + q^5 + 2*q^6 + 123*q^7 + 123*q^9 + O(q^10)),
  [q + 116*q^4 + 115*q^5 + 102*q^6 + 121*q^7 + 96*q^8 + 106*q^9 + O(q^10))], 4)
```

```
sage.modular.overconvergent.hecke_series.random_low_weight_bases(N, p, m, NN, weightbound)
```

Returns list of random integral bases of modular forms of level $N$ and (even) weight at most weightbound with coefficients reduced modulo $(p^m, q^{NN})$.

INPUT:

• $N$ – positive integer (level).
• $p$ – prime.
• $m, NN$ – positive integers.
• weightbound – (even) positive integer.

OUTPUT:

• list of lists of $q$-expansions modulo $(p^m, q^{NN})$.

EXAMPLES:

```
sage: from sage.modular.overconvergent.hecke_series import random_low_weight_bases
sage: S = random_low_weight_bases(3, 7, 2, 5, 6); S
# random
[[4 + 48*q + 46*q^2 + 48*q^3 + 42*q^4 + O(q^5)],
 [3 + 5*q + 45*q^2 + 22*q^3 + 22*q^4 + O(q^5),
  1 + 3*q + 27*q^2 + 27*q^3 + 23*q^4 + O(q^5),
  [2*q + 4*q^2 + 16*q^3 + 48*q^4 + O(q^5),
  2 + 6*q + q^2 + 3*q^3 + 43*q^4 + O(q^5),
  [1 + 2*q + 6*q^2 + 14*q^3 + 4*q^4 + O(q^5))]
 sage: S[0][0].parent()
Power Series Ring in q over Ring of integers modulo 49
 sage: S[0][0].prec()
5
```

```
sage.modular.overconvergent.hecke_series.random_new_basis_modp(N, p, k, LWModp,
 TotalBasisModp, elldash, bound)
```

4.12. Atkin/Hecke series for overconvergent modular forms 405
Returns a list of lists of lists \([j, a]\) encoding a choice of basis for the \(i\)th complementary space \(W_i\), as explained in the documentation for the function \texttt{complementary_spaces_modp()}.

**INPUT:**

- \(N\) – positive integer at least 2 and not divisible by \(p\) (level).
- \(p\) – prime at least 5.
- \(k\) – non-negative integer.
- \(\text{LWB}\) – list of \(q\)-expansions modulo \((p, q^{elldash})\).
- \(\text{TotalBasisModp}\) – matrix over \(\text{GF}(p)\).
- \(elldash\) – positive integer.
- \(\text{bound}\) – positive even integer (twice the length of the list \(\text{LWB}\)).

**OUTPUT:**

- A list of lists of lists \([j, a]\).

**Note:** As well as having a non-trivial return value, this function also modifies the input matrix \(\text{TotalBasisModp}\).

**EXAMPLES:**

```python
sage: from sage.modular.overconvergent.hecke_series import random_low_weight_bases, complementary_spaces_modp
sage: LWB = random_low_weight_bases(2, 5, 2, 3, 4)

sage: LWBModp = [ [f.change_ring(GF(5)) for f in x] for x in LWB]

sage: complementary_spaces_modp(2, 5, 2, 3, 4, LWBModp, 4)  # random, indirect
```

**sage.modular.overconvergent.hecke_series.random_solution\((B, K)\)**

Returns a random solution in non-negative integers to the equation \(a_1 + 2a_2 + 3a_3 + \ldots + Ba_B = K\), using a greedy algorithm.

Note that this is much faster than using \texttt{WeightedIntegerVectors.random_element()}.

**INPUT:**

- \(B, K\) – non-negative integers.

**OUTPUT:**

- list.

**EXAMPLES:**

```python
sage: from sage.modular.overconvergent.hecke_series import random_solution
sage: s = random_solution(5, 10)

sage: sum(s[i] * (i + 1) for i in range(5))
10

sage: S = set()

sage: while len(S) != 30:
    s = random_solution(5, 10)
    S.add(sum(s[i] * (i + 1) for i in range(5)))
```

(continues on next page)
....:     s = random_solution(5, 10)
....:     assert sum(s[i] * (i + 1) for i in range(5)) == 10
....:     S.add(tuple(s))

4.13 Module of supersingular points

The module of divisors on the modular curve $X_0(N)$ over $F_p$ supported at supersingular points.

EXAMPLES:

```python
sage: x = SupersingularModule(389)
sage: m = x.T(2).matrix()
sage: a = m.change_ring(GF(97))
sage: D = a.decomposition()
sage: D[:3]
[(Vector space of degree 33 and dimension 1 over Finite Field of size 97
  Basis matrix:
  [ 0 0 0 1 96 96 1 0 95 1 1 1 1 95 2 96 0 0 96 0 96 0 96 2 96 96 0 1 0
    -2 1 95 0], True),
  (Vector space of degree 33 and dimension 1 over Finite Field of size 97
  Basis matrix:
  [ 0 1 96 16 75 22 81 0 0 17 17 80 80 0 0 74 40 1 16 57 23 96 81 0 74 23 0 24 0
    ...0 73 0 0], True),
  (Vector space of degree 33 and dimension 1 over Finite Field of size 97
  Basis matrix:
  [ 0 1 96 90 90 7 7 0 0 91 6 6 91 0 0 91 0 13 7 0 6 84 90 0 6 91 0 90 0
    ...0 7 0 0], True)]
sage: len(D)
9
```

We compute a Hecke operator on a space of huge dimension!:

```python
sage: X = SupersingularModule(next_prime(10000))
sage: t = X.T(2).matrix()  # long time (21s on sage.math, 2011)
sage: t.nrows()  # long time
835
```

AUTHORS:

- William Stein
- David Kohel
- Iftikhar Burhanuddin

sage.modular.ssmod.ssm.m Phi2_quad(J3, ssJ1, ssJ2)

Return a certain quadratic polynomial over a finite field in indeterminate J3.

The roots of the polynomial along with ssJ1 are the neighboring/2-isogenous supersingular j-invariants of ssJ2.

INPUT:
Modular Forms, Release 10.0

- J3 – indeterminate of a univariate polynomial ring defined over a finite field with p^2 elements where p is a prime number
- ssJ2, ssJ2 – supersingular j-invariants over the finite field

OUTPUT:
- polynomial – defined over the finite field

EXAMPLES:
The following code snippet produces a factor of the modular polynomial \( \Phi_2(x, j_{\text{in}}) \), where \( j_{\text{in}} \) is a supersingular j-invariant defined over the finite field with 37^2 elements:

```sage
sage: F = GF(37^2, 'a')
sage: X = PolynomialRing(F, 'x').gen()
sage: j_in = supersingular_j(F)
sage: poly = sage.modular.ssmod.ssmod.Phi_polys(2, X, j_in)
sage: poly.roots()
[(8, 1), (27*a + 23, 1), (10*a + 20, 1)]
sage: sage.modular.ssmod.ssmod.Phi2_quad(X, F(8), j_in)
x^2 + 31*x + 31
```

Note: Given a root \((j1,j2)\) to the polynomial \( \Phi_{j2}(J1, J2) \), the pairs \((j2,j3)\) not equal to \((j2,j1)\) which solve \( \Phi_{j2}(j2, j3) \) are roots of the quadratic equation:

\[
J3^2 + (-j2^2 + 1488*j2 + (j1-162000))*J3 + (-j1+1488)*j2^2 + (1488*j1 + 40773375)*j2 + j1^2 - 162000*j1 + 8748000000
\]

This will be of use to extend the 2-isogeny graph, once the initial three roots are determined for \( \Phi_{j2}(J1, J2) \).

AUTHORS:
- David Kohel – kohel@maths.usyd.edu.au
- Iftikhar Burhanuddin – burhanud@usc.edu

sage.modular.ssmod.ssmod.Phi_polys(L, x, j)

Return a certain polynomial of degree \( L + 1 \) in the indeterminate \( x \) over a finite field.

The roots of the modular polynomial \( \Phi(L, x, j) \) are the \( L \)-isogenous supersingular j-invariants of \( j \).

INPUT:
- \( L \) – integer
- \( x \) – indeterminate of a univariate polynomial ring defined over a finite field with \( p^2 \) elements, where \( p \) is a prime number
- \( j \) – supersingular j-invariant over the finite field

OUTPUT:
- polynomial – defined over the finite field

EXAMPLES:
The following code snippet produces the modular polynomial \( \Phi_L(x, j_{\text{in}}) \), where \( j_{\text{in}} \) is a supersingular j-invariant defined over the finite field with 7^2 elements:
sage: F = GF(7^2, 'a')
sage: X = PolynomialRing(F, 'x').gen()
sage: j_in = supersingular_j(F)
sage: sage.modular.ssmod.ssmod.Phi_polys(2,X,j_in)
x^3 + 3*x^2 + 3*x + 1
sage: sage.modular.ssmod.ssmod.Phi_polys(3,X,j_in)
x^4 + 4*x^3 + 6*x^2 + 4*x + 1
sage: sage.modular.ssmod.ssmod.Phi_polys(5,X,j_in)
x^6 + 6*x^5 + x^4 + 6*x^3 + x^2 + 6*x + 1
sage: sage.modular.ssmod.ssmod.Phi_polys(7,X,j_in)
x^8 + x^7 + x + 1
sage: sage.modular.ssmod.ssmod.Phi_polys(11,X,j_in)
x^12 + 5*x^11 + 3*x^10 + 3*x^9 + 5*x^8 + x^7 + x^5 + 5*x^4 + 3*x^3 + 3*x^2 + 5*x + 1
sage: sage.modular.ssmod.ssmod.Phi_polys(13,X,j_in)
x^14 + 2*x^7 + 1

class sage.modular.ssmod.ssmod.SupersingularModule(prime=2, level=1, base_ring=Integer Ring)

Bases: HeckeModule_free_module

The module of supersingular points in a given characteristic, with given level structure.

The characteristic must not divide the level.

Note: Currently, only level 1 is implemented.

EXAMPLES:

sage: S = SupersingularModule(17)
sage: S
Module of supersingular points on X_0(1)/F_17 over Integer Ring
sage: S = SupersingularModule(16)
Traceback (most recent call last):
  ... ValueError: the argument prime must be a prime number
sage: S = SupersingularModule(prime=17, level=34)
Traceback (most recent call last):
  ... ValueError: the argument level must be coprime to the argument prime
sage: S = SupersingularModule(prime=17, level=5)
Traceback (most recent call last):
  ... NotImplementedError: supersingular modules of level > 1 not yet implemented

dimension()

Return the dimension of the space of modular forms of weight 2 and level equal to the level associated to self.

INPUT:

• self – SupersingularModule object

OUTPUT:

• integer – dimension, nonnegative

EXAMPLES:
```python
sage: S = SupersingularModule(7)
sage: S.dimension()
1
sage: S = SupersingularModule(15073)
sage: S.dimension()
1256
sage: S = SupersingularModule(83401)
sage: S.dimension()
6950
```

**Note:** The case of level > 1 has not yet been implemented.

**AUTHORS:**
- David Kohel – kohel@maths.usyd.edu.au
- Iftikhar Burhanuddin – burhanud@usc.edu

**free_module()**

**EXAMPLES:**

```python
sage: X = SupersingularModule(37)
sage: X.free_module()
Ambient free module of rank 3 over the principal ideal domain Integer Ring
```

This illustrates the fix at [github issue #4306:](https://github.com/)

```python
sage: X = SupersingularModule(389)
sage: X.basis()
((1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0),
 (0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0),
 (0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0),
 (0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0),
 (0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0),
 (0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0),
 (0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0),
 (0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0),
 (0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0),
 (0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0),
 (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0),
 (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0),
 (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0),
 (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0),
 (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0),
 (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0),
)```

(continues on next page)
hecke_matrix(L)

Return the $L^{th}$ Hecke matrix.

INPUT:

- self – SupersingularModule object
- L – integer, positive
OUTPUT:

- matrix – sparse integer matrix

EXAMPLES:

This example computes the action of the Hecke operator $T_2$ on the module of supersingular points on $X_0(1)/F_{37}$:

```python
sage: S = SupersingularModule(37)
sage: M = S.hecke_matrix(2)
sage: M
[1 1 1]
[1 0 2]
[1 2 0]
```

This example computes the action of the Hecke operator $T_3$ on the module of supersingular points on $X_0(1)/F_{67}$:

```python
sage: S = SupersingularModule(67)
sage: M = S.hecke_matrix(3)
sage: M
[0 0 0 0 2 2]
[0 0 1 1 1 1]
[0 1 2 0 1 1]
[0 1 2 0 1 0]
[1 1 0 1 0 1]
[1 1 1 0 1 0]
```

**Note:** The first list — list_j — returned by the supersingular_points function are the rows and column indexes of the above hecke matrices and its ordering should be kept in mind when interpreting these matrices.

AUTHORS:

- David Kohel – kohel@maths.usyd.edu.au
- Iftikhar Burhanuddin – burhanud@usc.edu

**level()**

This function returns the level associated to self.

INPUT:

- self – SupersingularModule object

OUTPUT:

- integer – the level, positive

EXAMPLES:

```python
sage: S = SupersingularModule(15073)
sage: S.level()
1
```

AUTHORS:

- David Kohel – kohel@maths.usyd.edu.au
• Iftikhar Burhanuddin – burhanud@usc.edu

prime()

Return the characteristic of the finite field associated to self.

INPUT:

• self – SupersingularModule object

OUTPUT:

• integer – characteristic, positive

EXAMPLES:

```
sage: S = SupersingularModule(19)
sage: S.prime()
19
```

AUTHORS:

• David Kohel – kohel@maths.usyd.edu.au
• Iftikhar Burhanuddin – burhanud@usc.edu

rank()

Return the dimension of the space of modular forms of weight 2 and level equal to the level associated to self.

INPUT:

• self – SupersingularModule object

OUTPUT:

• integer – dimension, nonnegative

EXAMPLES:

```
sage: S = SupersingularModule(7)
sage: S.dimension()
1

sage: S = SupersingularModule(15073)
sage: S.dimension()
1256

sage: S = SupersingularModule(83401)
sage: S.dimension()
6950
```

Note: The case of level > 1 has not yet been implemented.

AUTHORS:

• David Kohel – kohel@maths.usyd.edu.au
• Iftikhar Burhanuddin – burhanud@usc.edu

4.13. Module of supersingular points
supersingular_points()

Compute the supersingular j-invariants over the finite field associated to self.

INPUT:

• self – SupersingularModule object

OUTPUT:

• list_j, dict_j – list_j is the list of supersingular j-invariants, dict_j is a dictionary with these j-invariants as keys and their indexes as values. The latter is used to speed up j-invariant look-up. The indexes are based on the order of their discovery.

EXAMPLES:

The following examples calculate supersingular j-invariants over finite fields with characteristic 7, 11 and 37:

```
sage: S = SupersingularModule(7)
sage: S.supersingular_points()
([6], {6: 0})
sage: S = SupersingularModule(11)
sage: S.supersingular_points()[0]
[1, 0]
sage: S = SupersingularModule(37)
sage: S.supersingular_points()[0]
[8, 27*a + 23, 10*a + 20]
```

AUTHORS:

• David Kohel – kohel@maths.usyd.edu.au
• Iftikhar Burhanuddin – burhanud@usc.edu

upper_bound_on_elliptic_factors(p=None, ellmax=2)

Return an upper bound (provably correct) on the number of elliptic curves of conductor equal to the level of this supersingular module.

INPUT:

• p – (default: 997) prime to work modulo

ALGORITHM: Currently we only use $T_2$. Function will be extended to use more Hecke operators later. The prime $p$ is replaced by the smallest prime that does not divide the level.

EXAMPLES:

```
sage: SupersingularModule(37).upper_bound_on_elliptic_factors()
2
```

(There are 4 elliptic curves of conductor 37, but only 2 isogeny classes.)

weight()

Return the weight associated to self.

INPUT:

• self – SupersingularModule object

OUTPUT:
• integer – weight, positive

EXAMPLES:

```
sage: S = SupersingularModule(19)
sage: S.weight()
2
```

AUTHORS:

• David Kohel – kohel@maths.usyd.edu.au
• Iftikhar Burhanuddin – burhanud@usc.edu

`sage.modular.ssmod.ssmod.dimension_supersingular_module(prime, level=1)`

Return the dimension of the Supersingular module, which is equal to the dimension of the space of modular forms of weight 2 and conductor equal to `prime` times `level`.

INPUT:

• `prime` – integer, prime
• `level` – integer, positive

OUTPUT:

• `dimension` – integer, nonnegative

EXAMPLES:

The code below computes the dimensions of Supersingular modules with level=1 and prime = 7, 15073 and 83401:

```
sage: dimension_supersingular_module(7)
1
sage: dimension_supersingular_module(15073)
1256
sage: dimension_supersingular_module(83401)
6950
```

Note: The case of level > 1 has not been implemented yet.

AUTHORS:

• David Kohel – kohel@maths.usyd.edu.au
• Iftikhar Burhanuddin - burhanud@usc.edu

`sage.modular.ssmod.ssmod.supersingular_D(prime)`

Return a fundamental discriminant $D$ of an imaginary quadratic field, where the given prime does not split.

See Silverman’s Advanced Topics in the Arithmetic of Elliptic Curves, page 184, exercise 2.30(d).

INPUT:

• `prime` – integer, prime

OUTPUT:

• `D` – integer, negative

4.13. Module of supersingular points
EXAMPLES:
These examples return *supersingular discriminants* for 7, 15073 and 83401:

```plaintext
sage: supersingular_D(7)
-4
sage: supersingular_D(15073)
-15
sage: supersingular_D(83401)
-7
```

AUTHORS:
- David Kohel - kohel@maths.usyd.edu.au
- Iftikhar Burhanuddin - burhanud@usc.edu

`sage.modular.ssmod.ssmod.supersingular_j(FF)`
Return a supersingular j-invariant over the finite field FF.

**INPUT:**
- `FF` – finite field with $p^2$ elements, where $p$ is a prime number

**OUTPUT:**
- finite field element – a supersingular j-invariant defined over the finite field FF

**EXAMPLES:**
The following examples calculate supersingular j-invariants for a few finite fields:

```plaintext
sage: supersingular_j(GF(7^2, 'a'))
6
```

Observe that in this example the j-invariant is not defined over the prime field:

```plaintext
sage: supersingular_j(GF(15073^2, 'a'))
4443*a + 13964
sage: supersingular_j(GF(83401^2, 'a'))
67977
```

AUTHORS:
- David Kohel – kohel@maths.usyd.edu.au
- Iftikhar Burhanuddin – burhanud@usc.edu
4.14 Brandt modules

4.14.1 Introduction

The construction of Brandt modules provides us with a method to compute modular forms, as outlined in Pizer’s paper [Piz1980].

Given a prime number $p$ and a positive integer $M$ with $p \nmid M$, the Brandt module $B(p, M)$ is the free abelian group on right ideal classes of a quaternion order of level $pM$ in the quaternion algebra ramified precisely at the places $p$ and $\infty$. This Brandt module carries a natural Hecke action given by Brandt matrices. There exists a non-canonical Hecke algebra isomorphism between $B(p, M)$ and a certain subspace of $S_2(\Gamma_0(pM))$ containing the newforms.

4.14.2 Quaternion Algebras

A quaternion algebra over $\mathbb{Q}$ is a central simple algebra of dimension 4 over $\mathbb{Q}$. Such an algebra $A$ is said to be ramified at a place $v$ of $\mathbb{Q}$ if and only if $A \otimes \mathbb{Q}_v$ is a division algebra. Otherwise $A$ is said to be split at $v$.

$A = \text{QuaternionAlgebra}(p)$ returns the quaternion algebra $A$ over $\mathbb{Q}$ ramified precisely at the places $p$ and $\infty$.

$A = \text{QuaternionAlgebra}(a, b)$ returns the quaternion algebra $A$ over $\mathbb{Q}$ with basis $\{1, i, j, k\}$ such that $i^2 = a$, $j^2 = b$ and $ij = -ji = k$.

An order $R$ in a quaternion algebra $A$ over $\mathbb{Q}$ is a 4-dimensional lattice in $A$ which is also a subring containing the identity. A maximal order is one that is not properly contained in another order.

A particularly important kind of orders are those that have a level; see Definition 1.2 in [Piz1980]. This is a positive integer $N$ such that every prime that ramifies in $A$ divides $N$ to an odd power. The maximal orders are those that have level equal to the discriminant of $A$.

$R = A.\text{maximal_order}()$ returns a maximal order $R$ in the quaternion algebra $A$.

A right $\mathcal{O}$-ideal $I$ is a lattice in $A$ such that for every prime $p$ there exists $a_p \in A_p^*$ with $I_p = a_p\mathcal{O}_p$. Two right $\mathcal{O}$-ideals $I$ and $J$ are said to belong to the same class if $I = aJ$ for some $a \in A^*$. Left $\mathcal{O}$-ideals are defined in a similar fashion.

The right order of $I$ is the subring of $A$ consisting of elements $a$ with $Ia \subseteq I$.

4.14.3 Brandt Modules

$B = \text{BrandtModule}(p, M=1)$ returns the Brandt module associated to the prime number $p$ and the integer $M$, with $p$ not dividing $M$.

$A = B.\text{quaternion_algebra}()$ returns the quaternion algebra attached to $B$; this is the quaternion algebra over $\mathbb{Q}$ ramified exactly at $p$ and $\infty$.

$O = B.\text{order_of_level}_N()$ returns an order $\mathcal{O}$ of level $N = pM$ in $A$.

$B.\text{right_ideals}()$ returns a tuple of representatives for all right ideal classes of $\mathcal{O}$.

The implementation of this method is especially interesting. It depends on the construction of a Hecke module defined as a free abelian group on right ideal classes of a quaternion algebra with the following action:

$$T_n[I] = \sum_{\phi} [J]$$

where $(n, pM) = 1$ and the sum is over cyclic $\mathcal{O}$-module homomorphisms $\phi: I \to J$ of degree $n$ up to isomorphism of $J$. Equivalently one can sum over the inclusions of the submodules $J \to n^{-1}I$. The rough idea is to start with the trivial ideal class containing the order $\mathcal{O}$ itself. Using the method cyclic_submodules(self, I, q) one then repeatedly computes $T_q([\mathcal{O}])$ for some prime $q$ not dividing the level of $\mathcal{O}$ and tests for equivalence among the resulting
ideals. A theorem of Serre asserts that one gets a complete set of ideal class representatives after a finite number of repetitions.

One can prove that two ideals \(I\) and \(J\) are equivalent if and only if there exists an element \(\alpha \in IJ\) such that \(N(\alpha) = N(J)N(I)\).

\(\text{is\_equivalent}(I, J)\) returns true if \(I\) and \(J\) are equivalent. This method first compares the theta series of \(I\) and \(J\). If they are the same, it computes the theta series of the lattice \(I(J)\). It returns true if the \(n^{th}\) coefficient of this series is nonzero where \(n = N(J)N(I)\).

The theta series of a lattice \(L\) over the quaternion algebra \(A\) is defined as

\[
\theta_L(q) = \sum_{x \in L} q^{N(x)/N(L)}
\]

\(L.\text{theta\_series}(T, q)\) returns a power series representing \(\theta_L(q)\) up to a precision of \(O(q^{T+1})\).

### 4.14.4 Hecke Structure

The Hecke structure defined on the Brandt module is given by the Brandt matrices which can be computed using the definition of the Hecke operators given earlier.

\(\text{hecke\_matrix\_from\_defn}(\text{self}, n)\) returns the matrix of the \(n^{th}\) Hecke operator \(B_0(n)\) acting on \(\text{self}\), computed directly from the definition.

However, one can efficiently compute Brandt matrices using theta series. In fact, let \(\{I_1, \ldots, I_h\}\) be a set of right \(O\)-ideal class representatives. The \((i,j)\) entry in the Brandt matrix \(B_0(n)\) is the product of the \(n^{th}\) coefficient in the theta series of the lattice \(I_1 I_2\) and the first coefficient in the theta series of the lattice \(I_i I_j\).

\(\text{compute\_hecke\_matrix\_brandt}(\text{self}, n)\) returns the \(n^{th}\) Hecke matrix, computed using theta series.

**EXAMPLES:**

```python
sage: B = BrandtModule(23)
sage: B.maximal_order()
Order of Quaternion Algebra (-1, -23) with base ring Rational Field with basis (1/2 + 1/2*i + 1/2*j, 1/2*i + 1/2*k, j, k)
sage: B.right_ideals()
(Fractional ideal (2 + 2*j, 2*i + 2*k, 4*j, 4*k), Fractional ideal (2 + 2*j, 2*i + 6*k, 8*j, 8*k), Fractional ideal (2 + 10*j + 8*k, 2*i + 8*j + 6*k, 16*j, 16*k))
sage: B.hecke_matrix(2)
[1 2 0]
[1 1 1]
[0 3 0]
sage: B.brandt_series(3)
[1/4 + q + q^2 + O(q^3) 1/4 + q^2 + O(q^3) 1/4 + 0(q^3)]
[ 1/2 + 2*q^2 + O(q^3) 1/2 + q + q^2 + O(q^3) 1/2 + 3*q^2 + O(q^3)]
[ 1/6 + 0(q^3) 1/6 + q^2 + O(q^3) 1/6 + q + 0(q^3)]
```

**REFERENCES:**

- [Piz1980]
- [Koh2000]
4.14.5 Further Examples

We decompose a Brandt module over both $\mathbb{Z}$ and $\mathbb{Q}$.

```
sage: B = BrandtModule(43, base_ring=ZZ); B
Brandt module of dimension 4 of level 43 of weight 2 over Integer Ring
sage: D = B.decomposition()
sage: D
Subspace of dimension 1 of Brandt module of dimension 4 of level 43 of weight 2 over Integer Ring,
Subspace of dimension 1 of Brandt module of dimension 4 of level 43 of weight 2 over Integer Ring,
Subspace of dimension 2 of Brandt module of dimension 4 of level 43 of weight 2 over Integer Ring
sage: D[0].basis()
((0, 0, 1, -1),)
sage: D[1].basis()
((1, 2, 2, 2),)
sage: D[2].basis()
((1, 1, -1, -1), (0, 2, -1, -1))
sage: B = BrandtModule(43, base_ring=QQ); B
Brandt module of dimension 4 of level 43 of weight 2 over Rational Field
sage: B.decomposition()[2].basis()
((1, 0, -1/2, -1/2), (0, 1, -1/2, -1/2))
```

AUTHORS:
- Jon Bober
- Alia Hamieh
- Victoria de Quehen
- William Stein
- Gonzalo Tornaria

`sage.modular.quatalg.brandt.BrandtModule(N, M=1, weight=2, base_ring=Rational Field, use_cache=True)`

Return the Brandt module of given weight associated to the prime power $p^r$ and integer $M$, where $p$ and $M$ are coprime.

**INPUT:**
- $N$ – a product of primes with odd exponents
- $M$ – an integer coprime to $q$ (default: 1)
- `weight` – an integer that is at least 2 (default: 2)
- `base_ring` – the base ring (default: QQ)
- `use_cache` – whether to use the cache (default: True)

**OUTPUT:**
a Brandt module

**EXAMPLES:**
sage: BrandtModule(17)
Brandt module of dimension 2 of level 17 of weight 2 over Rational Field
sage: BrandtModule(17,15)
Brandt module of dimension 32 of level 17*15 of weight 2 over Rational Field
sage: BrandtModule(3,7)
Brandt module of dimension 2 of level 3*7 of weight 2 over Rational Field
sage: BrandtModule(3,weight=2)
Brandt module of dimension 1 of level 3 of weight 2 over Rational Field
sage: BrandtModule(11, base_ring=ZZ)
Brandt module of dimension 2 of level 11 of weight 2 over Integer Ring
sage: BrandtModule(11, base_ring=QQbar)
Brandt module of dimension 2 of level 11 of weight 2 over Algebraic Field

The use_cache option determines whether the Brandt module returned by this function is cached:

sage: BrandtModule(37) is BrandtModule(37)
True
sage: BrandtModule(37,use_cache=False) is BrandtModule(37,use_cache=False)
False

class sage.modular.quatalg.brandt.BrandtModuleElement(parent, x)

Bases: HeckeModuleElement

EXAMPLES:

sage: B = BrandtModule(37)
sage: x = B([1,2,3]); x
(1, 2, 3)
sage: parent(x)
Brandt module of dimension 3 of level 37 of weight 2 over Rational Field

monodromy_pairing(x)

Return the monodromy pairing of self and x.

EXAMPLES:

sage: B = BrandtModule(5,13)
sage: B.monodromy_weights()
(1, 3, 1, 1, 1, 3)
sage: (B.0 + B.1).monodromy_pairing(B.0 + B.1)
4

class sage.modular.quatalg.brandt.BrandtModule_class(N, M, weight, base_ring)

Bases: AmbientHeckeModule

A Brandt module.

EXAMPLES:

sage: BrandtModule(3, 10)
Brandt module of dimension 4 of level 3*10 of weight 2 over Rational Field

Element

alias of BrandtModuleElement
M()  
Return the auxiliary level (prime to $p$ part) of the quaternion order used to compute this Brandt module.

EXAMPLES:

```sage
sage: BrandtModule(7,5,2,ZZ).M()
5
```

N()  
Return ramification level $N$.

EXAMPLES:

```sage
sage: BrandtModule(7,5,2,ZZ).N()
7
```

brandt_series(prec, var='q')  
Return matrix of power series $\sum T_n q^n$ to the given precision.

Note that the Hecke operators in this series are always over $\mathbb{Q}$, even if the base ring of this Brandt module is not $\mathbb{Q}$.

INPUT:
- `prec` – positive integer
- `var` – string (default: `q`)

OUTPUT:
matrix of power series with coefficients in $\mathbb{Q}$

EXAMPLES:

```sage
sage: B = BrandtModule(11)
sage: B.brandt_series(2)
[1/4 + q + O(q^2) 1/4 + O(q^2)]
[ 1/6 + O(q^2) 1/6 + q + O(q^2)]
sage: B.brandt_series(5)
[1/4 + q + q^2 + 2*q^3 + 5*q^4 + O(q^5) 1/4 + 3*q^2 + 3*q^3 + 3*q^4 + O(q^5)]
[ 1/6 + 2*q^2 + 2*q^3 + 2*q^4 + O(q^5) 1/6 + q + q^3 + 4*q^4 + O(q^5)]
```

Asking for a smaller precision works:

```sage
sage: B.brandt_series(3)
[1/4 + q + q^2 + O(q^3) 1/4 + 3*q^2 + O(q^3)]
[ 1/6 + 2*q^2 + O(q^3) 1/6 + q + O(q^3)]
sage: B.brandt_series(3, var='t')
[1/4 + t + t^2 + O(t^3) 1/4 + 3*t^2 + O(t^3)]
[ 1/6 + 2*t^2 + O(t^3) 1/6 + t + O(t^3)]
```

classical()  
The character of this space.

Always trivial.

EXAMPLES:
cyclic_submodules(I, p)

Return a list of rescaled versions of the fractional right ideals \( J \) such that \( J \) contains \( I \) and the quotient has group structure the product of two cyclic groups of order \( p \).

We emphasize again that \( J \) is rescaled to be integral.

INPUT:

- \( I \) – ideal \( I \) in \( R = \text{self.order_of_level_N()} \)
- \( p \) – prime \( p \) coprime to \( \text{self.level()} \)

OUTPUT:

list of the \( p + 1 \) fractional right \( R \)-ideals that contain \( I \) such that \( J/I \) is \( \text{GF}(p) \times \text{GF}(p) \).

EXAMPLES:

```python
sage: B = BrandtModule(11)
sage: I = B.order_of_level_N().unit_ideal()
sage: B.cyclic_submodules(I, 2)
[Fractional ideal (1/2 + 3/2*j + k, 1/2*i + j + 1/2*k, 2*j, 2*k),
 Fractional ideal (1/2 + 1/2*i + 1/2*j + 1/2*k, i + k, j + k, 2*k),
 Fractional ideal (1/2 + 1/2*j + k, 1/2*i + j + 3/2*k, 2*j, 2*k)]
```

```python
sage: B.cyclic_submodules(I, 3)
[Fractional ideal (1/2 + 1/2*j, 1/2*i + 5/2*k, 3*j, 3*k),
 Fractional ideal (1/2 + 3/2*j + 2*k, 1/2*i + 2*j + 3/2*k, 3*j, 3*k),
 Fractional ideal (1/2 + 3/2*j + k, 1/2*i + j + 3/2*k, 3*j, 3*k),
 Fractional ideal (1/2 + 5/2*j, 1/2*i + 1/2*k, 3*j, 3*k)]
```

```python
sage: B.cyclic_submodules(I, 11)
Traceback (most recent call last):
  ... ValueError: p must be coprime to the level
```

eisenstein_subspace()

Return the 1-dimensional subspace of \( \text{self} \) on which the Hecke operators \( T_p \) act as \( p + 1 \) for \( p \) coprime to the level.

**Note:** This function assumes that the base field has characteristic 0.

EXAMPLES:

```python
sage: B = BrandtModule(11); B.eisenstein_subspace()
Subspace of dimension 1 of Brandt module of dimension 2 of level 11 of weight 2 over Rational Field
```

```python
sage: B.eisenstein_subspace() \is B.eisenstein_subspace()
True
```

```python
sage: BrandtModule(3,11).eisenstein_subspace().basis()
((1, 1),)
```

```python
sage: BrandtModule(7,10).eisenstein_subspace().basis()
((1, 1, 1, 1/2, 1, 1, 1/2, 1, 1, 1),)
```

(continues on next page)
```python
sage: BrandtModule(7,10,base_ring=ZZ).eisenstein_subspace().basis()
((2, 2, 2, 1, 2, 1, 2, 2),)
```

**free_module()**

Return the underlying free module of the Brandt module.

**EXAMPLES:**

```python
sage: B = BrandtModule(10007,389)
sage: B.free_module()
Vector space of dimension 325196 over Rational Field
```

**hecke_matrix**(\(n, \text{algorithm}='\text{default}', \text{sparse}=False, B=None\))

Return the matrix of the \(n\)-th Hecke operator.

**INPUT:**

- \(n\) – integer
- \text{algorithm} – string (default: ‘default’)
  - ‘default’ – let Sage guess which algorithm is best
  - ‘direct’ – use cyclic subideals (generally much better when you want few Hecke operators and the dimension is very large); uses ‘theta’ if \(n\) divides the level.
  - ‘brandt’ – use Brandt matrices (generally much better when you want many Hecke operators and the dimension is very small; bad when the dimension is large)
- \text{sparse} – bool (default: False)
- \(B\) – integer or None (default: None); in direct algorithm, use theta series to this precision as an initial check for equality of ideal classes.

**EXAMPLES:**

```python
sage: B = BrandtModule(3,7); B.hecke_matrix(2)
[[0 3]
 [1 2]]
sage: B.hecke_matrix(5, algorithm='brandt')
[[0 6]
 [2 4]]
sage: t = B.hecke_matrix(11, algorithm='brandt', sparse=True); t
[[ 6  6]
 [ 2 10]]
sage: type(t)
<class 'sage.matrix.matrix_rational_sparse.Matrix_rational_sparse'>
sage: B.hecke_matrix(19, algorithm='direct', B=2)
[[ 8 12]
 [ 4 16]]
```

**is_cuspidal()**

Return whether \text{self} is cuspidal, i.e. has no Eisenstein part.

**EXAMPLES:**
sage: B = BrandtModule(3, 4)
sage: B.is_cuspidal()
False
sage: B.eisenstein_subspace()
Brandt module of dimension 1 of level 3*4 of weight 2 over Rational Field

**maximal_order()**

Return a maximal order in the quaternion algebra associated to this Brandt module.

EXAMPLES:

sage: BrandtModule(17).maximal_order()
Order of Quaternion Algebra (-3, -17) with base ring Rational Field with basis
→(1/2 + 1/2*i, 1/2*j - 1/2*k, -1/3*i + 1/3*k, -k)
sage: BrandtModule(17).maximal_order() is BrandtModule(17).maximal_order()
True

**monodromy_weights()**

Return the weights for the monodromy pairing on this Brandt module.

The weights are associated to each ideal class in our fixed choice of basis. The weight of an ideal class \([I]\) is half the number of units of the right order \(I\).

Note: The base ring must be \(\mathbb{Q}\) or \(\mathbb{Z}\).

EXAMPLES:

sage: BrandtModule(11).monodromy_weights()
(2, 3)
sage: BrandtModule(37).monodromy_weights()
(1, 1, 1)
sage: BrandtModule(43).monodromy_weights()
(2, 1, 1, 1)
sage: BrandtModule(7,10).monodromy_weights()
(1, 1, 1, 2, 1, 1, 2, 1, 1, 1)
sage: BrandtModule(5,13).monodromy_weights()
(1, 3, 1, 1, 1, 3)
sage: BrandtModule(2).monodromy_weights()
(12,)
sage: BrandtModule(2,7).monodromy_weights()
(3, 3)

**order_of_level_N()**

Return an order of level \(N = p^{2r+1}M\) in the quaternion algebra.

EXAMPLES:

sage: BrandtModule(7).order_of_level_N()
Order of Quaternion Algebra (-1, -7) with base ring Rational Field with basis
→(1/2 + 1/2*i, 1/2*j + 1/2*k, j, k)
sage: BrandtModule(7,13).order_of_level_N()
Order of Quaternion Algebra (-1, -7) with base ring Rational Field with basis
→(1/2 + 1/2*i + 12*k, 1/2*i + 9/2*k, j + 11*k, 13*k)
sage: BrandtModule(7, 3*17).order_of_level_N()
Order of Quaternion Algebra (-1, -7) with basis
→(1/2 + 1/2*j + 35*k, 1/2*i + 65/2*k, j + 19*k, 51*k)

quaternion_algebra()
Return the quaternion algebra $\mathcal{A}$ over $\mathbb{Q}$ ramified precisely at $p$ and infinity used to compute this Brandt module.

EXAMPLES:

sage: BrandtModule(997).quaternion_algebra()
Quaternion Algebra (-2, -997) with base ring Rational Field
sage: BrandtModule(2).quaternion_algebra()
Quaternion Algebra (-1, -1) with base ring Rational Field
sage: BrandtModule(3).quaternion_algebra()
Quaternion Algebra (-1, -3) with base ring Rational Field
sage: BrandtModule(5).quaternion_algebra()
Quaternion Algebra (-2, -5) with base ring Rational Field
sage: BrandtModule(17).quaternion_algebra()
Quaternion Algebra (-3, -17) with base ring Rational Field

right_ideals($B$=None)
Return sorted tuple of representatives for the equivalence classes of right ideals in self.

OUTPUT:
sorted tuple of fractional ideals

EXAMPLES:

sage: B = BrandtModule(23)
sage: B.right_ideals()
(Fractional ideal (2 + 2*j, 2*i + 2*k, 4*j, 4*k),
 Fractional ideal (2 + 2*j, 2*i + 6*k, 8*j, 8*k),
 Fractional ideal (2 + 10*j + 8*k, 2*i + 8*j + 6*k, 16*j, 16*k))

4.14. Brandt modules
\[\rightarrow 2^*k, A(3)\]  
\[1/2 + 1/6*i + 1/3*k, 1/3*i + 2/3*k, 1/2*j + 1/2*k, k\]  
sage: sage.modular.quatalg.brandt.basis_for_left_ideal(B.maximal_order(), [3*(i+j), \rightarrow 3^*(i-j), 6^*k, A(3)])  
\[3/2 + 1/2*i + k, i + 2*k, 3/2*j + 3/2*k, 3^*k\]  

sage.modular.quatalg.brandt.benchmark_magma(levels, silent=False)

INPUT:
- levels – list of pairs \((p, M)\) where \(p\) is a prime not dividing \(M\)
- silent – bool, default False; if True suppress printing during computation

OUTPUT:
list of 4-tuples (`magma`, p, M, tm), where tm is the CPU time in seconds to compute T2 using Magma

EXAMPLES:

```python
sage: a = sage.modular.quatalg.brandt.benchmark_magma([(11,1), (37,1), (43,1), (97, ˓→1)])  # optional - magma
('magma', 11, 1, ...)
('magma', 37, 1, ...)
('magma', 43, 1, ...)
('magma', 97, 1, ...

sage: a = sage.modular.quatalg.brandt.benchmark_magma([(11,2), (37,2), (43,2), (97, ˓→2)])  # optional - magma
('magma', 11, 2, ...)
('magma', 37, 2, ...)
('magma', 43, 2, ...)
('magma', 97, 2, ...)
```

sage.modular.quatalg.brandt.benchmark_sage(levels, silent=False)

INPUT:
- levels – list of pairs \((p, M)\) where \(p\) is a prime not dividing \(M\)
- silent – bool, default False; if True suppress printing during computation

OUTPUT:
list of 4-tuples (`sage`, p, M, tm), where tm is the CPU time in seconds to compute T2 using Sage

EXAMPLES:

```python
sage: a = sage.modular.quatalg.brandt.benchmark_sage([(11,1), (37,1), (43,1), (97, ˓→1)])
('sage', 11, 1, ...)
('sage', 37, 1, ...)
('sage', 43, 1, ...)
('sage', 97, 1, ...

sage: a = sage.modular.quatalg.brandt.benchmark_sage([(11,2), (37,2), (43,2), (97, ˓→2)])
('sage', 11, 2, ...)
('sage', 37, 2, ...)
('sage', 43, 2, ...)
('sage', 97, 2, ...)
```
sage.modular.quatalg.brandt.class_number(p, r, M)

Return the class number of an order of level \( N = p^r M \) in the quaternion algebra over \( \mathbb{Q} \) ramified precisely at \( p \) and infinity.

This is an implementation of Theorem 1.12 of [Piz1980].

INPUT:

- \( p \) – a prime
- \( r \) – an odd positive integer (default: 1)
- \( M \) – an integer coprime to \( q \) (default: 1)

OUTPUT:

Integer

EXAMPLES:

```
sage: sage.modular.quatalg.brandt.class_number(389,1,1)
33
sage: sage.modular.quatalg.brandt.class_number(389,1,2) # TODO -- right?
97
sage: sage.modular.quatalg.brandt.class_number(389,3,1) # TODO -- right?
4892713
```

sage.modular.quatalg.brandt.maximal_order(A)

Return a maximal order in the quaternion algebra ramified at \( p \) and infinity.

This is an implementation of Proposition 5.2 of [Piz1980].

INPUT:

- \( A \) – quaternion algebra ramified precisely at \( p \) and infinity

OUTPUT:

a maximal order in \( A \)

EXAMPLES:

```
sage: A = BrandtModule(17).quaternion_algebra()
sage: sage.modular.quatalg.brandt.maximal_order(A)
Order of Quaternion Algebra (-3, -17) with base ring Rational Field with basis (1/2 + 1/2*i, 1/2*j - 1/2*k, -1/3*i + 1/3*k, -k)
sage: A = QuaternionAlgebra(17,names='i,j,k')
sage: A.maximal_order()
Order of Quaternion Algebra (-3, -17) with base ring Rational Field with basis (1/2 + 1/2*i, 1/2*j - 1/2*k, -1/3*i + 1/3*k, -k)
```

sage.modular.quatalg.brandt.quaternion_order_with_given_level(A, level)

Return an order in the quaternion algebra \( A \) with given level.

This is implemented only when the base field is the rational numbers.

INPUT:

- \( \text{level} \) – The level of the order to be returned. Currently this is only implemented when the level is divisible by at most one power of a prime that ramifies in this quaternion algebra.

EXAMPLES:
sage: from sage.modular.quatalg.brandt import quaternion_order_with_given_level;
    — maximal_order
sage: A.<i,j,k> = QuaternionAlgebra(5)
sage: level = 2 * 5 * 17
sage: O = quaternion_order_with_given_level(A, level)
→ sage: M = maximal_order(A)
→ sage: L = O.free_module()
→ sage: N = M.free_module()
→ sage: L.index_in(N) == level/5  #check that the order has the right index in the
    — maximal order
True

sage.modular.quatalg.brandt.right_order(R, basis)
Given a basis for a left ideal $I$, return the right order in the quaternion order $R$ of elements $x$ such that $Ix$ is contained in $I$.

INPUT:

- $R$ – order in quaternion algebra
- basis – basis for an ideal $I$

OUTPUT:

order in quaternion algebra

EXAMPLES:

We do a consistency check with the ideal equal to a maximal order:

\[
\text{sage: } B = \text{BrandtModule}(17); \text{basis} = \text{sage.modular.quatalg.brandt.basis_for_left_ideal}(B.\text{maximal_order}(), B.\text{maximal_order}().\text{basis}())
\]
\[
\text{sage: } \text{sage.modular.quatalg.brandt.right_order}(B.\text{maximal_order}(), \text{basis})
\]
Order of Quaternion Algebra (-3, -17) with base ring Rational Field with basis (1/2 + 1/6*i + 1/3*k, 1/3*i + 2/3*k, 1/2*j + 1/2*k, k)

\[
\text{sage: } \text{basis} = [1/2 + 1/6*i + 1/3*k, 1/3*i + 2/3*k, 1/2*j + 1/2*k, k]
\]

\[
\text{sage: } B = \text{BrandtModule}(17); A = B.\text{quaternion_algebra}(); i,j,k = A.\text{gens}()
\]
\[
\text{sage: } \text{basis} = \text{sage.modular.quatalg.brandt.basis_for_left_ideal}(B.\text{maximal_order}(), \text{[i*j-j]})
\]
\[
\text{sage: } \text{sage.modular.quatalg.brandt.right_order}(B.\text{maximal_order}(), \text{basis})
\]
Order of Quaternion Algebra (-3, -17) with base ring Rational Field with basis (1/2 + 1/6*i + 1/3*k, 1/3*i + 2/3*k, 1/2*j + 1/2*k, k)

\[
\text{sage: } \text{basis} = [1/2 + 1/6*i + 1/3*k, 1/3*i + 2/3*k, 1/2*j + 1/2*k, k]
\]

4.15 The set $\mathbb{P}^1(K)$ of cusps of a number field $K$

AUTHORS:


EXAMPLES:

The space of cusps over a number field $k$: 

428 Chapter 4. Miscellaneous Modules (to be sorted)
Define a cusp over a number field:

```python
sage: NFCusp(k, a, 2/(a+1))
Cusp [a - 5: 2] of Number Field in a with defining polynomial x^2 + 5
sage: NFCusp(k, oo)
Cusp Infinity of Number Field in a with defining polynomial x^2 + 5
```

Different operations with cusps over a number field:

```python
sage: alpha = NFCusp(k, 3, 1/a + 2); alpha
Cusp [a + 10: 7] of Number Field in a with defining polynomial x^2 + 5
sage: alpha.numerator()
a + 10
sage: alpha.denominator()
7
sage: alpha.ideal()
Fractional ideal (7, a + 3)
sage: M = alpha.ABmatrix(); M
# random
[[a + 10, 2*a + 6, 7, a + 5]]
```

Check Gamma0(N)-equivalence of cusps:

```python
sage: N = k.ideal(3)
sage: alpha = NFCusp(k, 3, a + 1)
sage: beta = kCusps((2, a - 3))
sage: alpha.is_Gamma0_equivalent(beta, N)
True
```

Obtain transformation matrix for equivalent cusps:

```python
sage: t, M = alpha.is_Gamma0_equivalent(beta, N, Transformation=True)
sage: M[2] in N
True
True
sage: alpha.apply(M) == beta
True
```

List representatives for Gamma_0(N) - equivalence classes of cusps:

```python
sage: Gamma0_NFCusps(N)
[Cusp [0: 1] of Number Field in a with defining polynomial x^2 + 5,
 Cusp [1: 3] of Number Field in a with defining polynomial x^2 + 5,
 ...]
```

4.15. The set $\mathbb{P}^1(K)$ of cusps of a number field $K$
sage.modular.cusps_nf.Gamma0_NFCusps(N)

Return a list of inequivalent cusps for $\Gamma_0(N)$, i.e., a set of representatives for the orbits of $\text{self}$ on $\mathbb{P}^1(k)$.

**INPUT:**
- $N$ – an integral ideal of the number field $k$ (the level).

**OUTPUT:**
A list of inequivalent number field cusps.

**EXAMPLES:**

```python
sage: k.<a> = NumberField(x^2 + 5)
sage: N = k.ideal(3)
sage: L = Gamma0_NFCusps(N)
```

The cusps in the list are inequivalent:

```python
sage: any(L[i].is_Gamma0_equivalent(L[j], N) for i in range(len(L)) for j in range(len(L)) if i < j)
False
```

We test that we obtain the right number of orbits:

```python
sage: from sage.modular.cusps_nf import number_of_Gamma0_NFCusps
sage: len(L) == number_of_Gamma0_NFCusps(N)
True
```

Another example:

```python
sage: k.<a> = NumberField(x^4 - x^3 -21*x^2 + 17*x + 133)
sage: N = k.ideal(5)
sage: from sage.modular.cusps_nf import number_of_Gamma0_NFCusps
sage: len(Gamma0_NFCusps(N)) == number_of_Gamma0_NFCusps(N)  # long time (over 1 sec)
True
```

class sage.modular.cusps_nf.NFCusp(number_field, a, b=None, parent=None, lreps=None)

Create a number field cusp, i.e., an element of $\mathbb{P}^1(k)$.

A cusp on a number field is either an element of the field or infinity, i.e., an element of the projective line over the number field. It is stored as a pair $(a,b)$, where $a, b$ are integral elements of the number field.

**INPUT:**
- number_field – the number field over which the cusp is defined.
- a – it can be a number field element (integral or not), or a number field cusp.
- b – (optional) when present, it must be either Infinity or coercible to an element of the number field.
- lreps – (optional) a list of chosen representatives for all the ideal classes of the field. When given, the representative of the cusp will be changed so its associated ideal is one of the ideals in the list.

**OUTPUT:**
[a, b] – a number field cusp.

**EXAMPLES:**
```python
sage: k.<a> = NumberField(x^2 + 5)
sage: NFCusp(k, a, 2)
Cusp [a: 2] of Number Field in a with defining polynomial x^2 + 5
sage: NFCusp(k, (a,2))
Cusp [a: 2] of Number Field in a with defining polynomial x^2 + 5
sage: NFCusp(k, a, 2/(a+1))
Cusp [a - 5: 2] of Number Field in a with defining polynomial x^2 + 5

Cusp Infinity:

sage: NFCusp(k, 0)
Cusp [0: 1] of Number Field in a with defining polynomial x^2 + 5
sage: NFCusp(k, oo)
Cusp Infinity of Number Field in a with defining polynomial x^2 + 5

Saving and loading works:

sage: alpha = NFCusp(k, a, 2/(a+1))
sage: loads(dumps(alpha))==alpha
True

Some tests:

sage: I*I
-1
sage: NFCusp(k, I)
Traceback (most recent call last):
  ...
TypeError: unable to convert I to a cusp of the number field

sage: NFCusp(k, oo, oo)
Traceback (most recent call last):
  ...
TypeError: unable to convert (+Infinity, +Infinity) to a cusp of the number field

sage: NFCusp(k, 0, 0)
Traceback (most recent call last):
  ...
TypeError: unable to convert (0, 0) to a cusp of the number field

sage: NFCusp(k, "a + 2", a)
Cusp [-2*a + 5: 5] of Number Field in a with defining polynomial x^2 + 5

sage: NFCusp(k, NFCusp(k, oo))
Cusp Infinity of Number Field in a with defining polynomial x^2 + 5
sage: c = NFCusp(k, 3, 2*a)
sage: NFCusp(k, c, a + 1)
Cusp [-a - 5: 20] of Number Field in a with defining polynomial x^2 + 5
```

4.15. The set $\mathbb{P}^1(K)$ of cusps of a number field $K$
Modular Forms, Release 10.0

sage: L.<b> = NumberField(x^2 + 2)
sage: NFCusp(L, c)
Traceback (most recent call last):
...
ValueError: Cannot coerce cusps from one field to another

ABmatrix()

Return AB-matrix associated to the cusp self.

Given R a Dedekind domain and A, B ideals of R in inverse classes, an AB-matrix is a matrix realizing the isomorphism between R+R and A+B. An AB-matrix associated to a cusp [a1: a2] is an AB-matrix with A the ideal associated to the cusp (A=<a1, a2>) and first column given by the coefficients of the cusp.

EXAMPLES:

sage: k.<a> = NumberField(x^3 + 11)
sage: alpha = NFCusp(k, oo)
sage: alpha.ABmatrix()
[1, 0, 0, 1]
sage: alpha = NFCusp(k, 0)
sage: alpha.ABmatrix()
[0, -1, 1, 0]

Note that the AB-matrix associated to a cusp is not unique, and the output of the ABmatrix function may change.

sage: alpha = NFCusp(k, 3/2, a-1)
sage: M = alpha.ABmatrix()
sage: M
[-a^2 - a - 1, -3*a - 7, 8, -2*a^2 - 3*a + 4]
True

An AB-matrix associated to a cusp alpha will send Infinity to alpha:

sage: alpha = NFCusp(k, 3, a-1)
sage: M = alpha.ABmatrix()
sage: (k.ideal(M[1], M[3])*alpha.ideal()).is_principal()
True
True
sage: NFCusp(k, oo).apply(M) == alpha
True

apply(g)

Return g(self), where g is a 2x2 matrix, which we view as a linear fractional transformation.

INPUT:

• g – a list of integral elements [a, b, c, d] that are the entries of a 2x2 matrix.

OUTPUT:

A number field cusp, obtained by the action of g on the cusp self.
EXAMPLES:

```python
sage: k.<a> = NumberField(x^2 + 23)
sage: beta = NFCusp(k, 0, 1)
sage: beta.apply([0, -1, 1, 0])
Cusp Infinity of Number Field in a with defining polynomial x^2 + 23
sage: beta.apply([1, a, 0, 1])
Cusp [a: 1] of Number Field in a with defining polynomial x^2 + 23
```

**denominator()**

Return the denominator of the cusp self.

EXAMPLES:

```python
sage: k.<a> = NumberField(x^2 + 1)
sage: c = NFCusp(k, a, 2)
sage: c.denominator()
2
sage: d = NFCusp(k, 1, a + 1); d
Cusp [1: a + 1] of Number Field in a with defining polynomial x^2 + 1
sage: d.denominator()
a + 1
sage: NFCusp(k, oo).denominator()
0
```

**ideal()**

Return the ideal associated to the cusp self.

EXAMPLES:

```python
sage: k.<a> = NumberField(x^2 + 23)
sage: alpha = NFCusp(k, 3, a-1)
sage: alpha.ideal()
Fractional ideal (3, 1/2*a - 1/2)
sage: NFCusp(k, oo).ideal()
Fractional ideal (1)
```

**is_gamma0_equivalent**(*other, N, Transformation=False*)

Check if cusps self and other are \(\Gamma_0(N)\)-equivalent.

INPUT:

- other – a number field cusp or a list of two number field elements which define a cusp.
- N – an ideal of the number field (level)

OUTPUT:

- bool – True if the cusps are equivalent.
- a transformation matrix – (if Transformation=True) a list of integral elements \([a, b, c, d]\) which are the entries of a 2x2 matrix \(M\) in \(\Gamma_0(N)\) such that \(M \cdot \text{self} = \text{other}\) if other and self are \(\Gamma_0(N)\)-equivalent. If self and other are not equivalent it returns zero.

EXAMPLES:

```python
sage: K.<a> = NumberField(x^3-10)
sage: N = K.ideal(a-1)
```

4.15. The set \(P^1(K)\) of cusps of a number field \(K\)
sage: alpha = NFCusp(K, 0)
sage: beta = NFCusp(K, oo)
sage: alpha.is_Gamma0_equivalent(beta, N)
False
sage: alpha.is_Gamma0_equivalent(beta, K.ideal(1))
True
sage: b, M = alpha.is_Gamma0_equivalent(beta, K.ideal(1), Transformation=True)
sage: alpha.apply(M)
Cusp Infinity of Number Field in a with defining polynomial x^3 - 10

sage: k.<a> = NumberField(x^2 + 23)
sage: N = k.ideal(3)
sage: alpha1 = NFCusp(k, a+1, 4)
sage: alpha2 = NFCusp(k, a-8, 29)
sage: alpha1.is_Gamma0_equivalent(alpha2, N)
True
sage: b, M = alpha1.is_Gamma0_equivalent(alpha2, N, Transformation=True)
sage: alpha1.apply(M) == alpha2
True
sage: M[2] in N
True

is_infinity()

Return True if this is the cusp infinity.

EXAMPLES:

sage: k.<a> = NumberField(x^2 + 1)
sage: NFCusp(k, a, 2).is_infinity()
False
sage: NFCusp(k, 2, 0).is_infinity()
True
sage: NFCusp(k, oo).is_infinity()
True

number_field()

Return the number field of definition of the cusp self.

EXAMPLES:

sage: k.<a> = NumberField(x^2 + 2)
sage: alpha = NFCusp(k, 1, a + 1)
sage: alpha.number_field()
Number Field in a with defining polynomial x^2 + 2

numerator()

Return the numerator of the cusp self.

EXAMPLES:

sage: k.<a> = NumberField(x^2 + 1)
sage: c = NFCusp(k, a, 2)
sage: c.numerator()
```python
sage: d = NFCusp(k, 1, a)
sage: d.numerator()
1
sage: NFCusp(k, oo).numerator()
1
```

The set of cusps of a number field $K$, i.e. $\mathbb{P}^1(K)$.

**INPUT:**
- `number_field` – a number field

**OUTPUT:**
The set of cusps over the given number field.

**EXAMPLES:**
```python
sage: k.<a> = NumberField(x^2 + 5)
sage: kCusps = NFCusps(k); kCusps
Set of all cusps of Number Field in a with defining polynomial x^2 + 5
sage: kCusps is NFCusps(k)
True
```

Saving and loading works:
```python
sage: loads(kCusps.dumps()) == kCusps
True
```

```python
class NFCuspsSpace(number_field)

Bases: UniqueRepresentation, Parent

The set of cusps of a number field. See NFCusps for full documentation.

**EXAMPLES:**
```python
sage: k.<a> = NumberField(x^2 + 5)
sage: kCusps = NFCusps(k); kCusps
Set of all cusps of Number Field in a with defining polynomial x^2 + 5
```

**number_field()**

Return the number field that this set of cusps is attached to.

**EXAMPLES:**
```python
sage: k.<a> = NumberField(x^2 + 1)
sage: kCusps = NFCusps(k)
sage: kCusps.number_field()
Number Field in a with defining polynomial x^2 + 1
```

**zero()**

Return the zero cusp.

4.15. The set $\mathbb{P}^1(K)$ of cusps of a number field $K$
Note: This method just exists to make some general algorithms work. It is not intended that the returned cusp is an additive neutral element.

EXAMPLES:

```
sage: k.<a> = NumberField(x^2 + 5)
sage: kCusps = NFCusps(k)
sage: kCusps.zero()
Cusp [0: 1] of Number Field in a with defining polynomial x^2 + 5
```

`sage.modular.cusps_nf.NFCusps_ideal_reps_for_levelN(N, nlists=1)`

Return a list of lists (nlists different lists) of prime ideals, coprime to \( N \), representing every ideal class of the number field.

INPUT:

• \( N \) – number field ideal.

• nlists – optional (default 1). The number of lists of prime ideals we want.

OUTPUT:

A list of lists of ideals representatives of the ideal classes, all coprime to \( N \), representing every ideal.

EXAMPLES:

```
sage: k.<a> = NumberField(x^3 + 11)
sage: N = k.ideal(5, a + 1)
sage: from sage.modular.cusps_nf import NFCusps_ideal_reps_for_levelN
sage: NFCusps_ideal_reps_for_levelN(N)
[(Fractional ideal (1), Fractional ideal (2, a + 1))]
sage: L = NFCusps_ideal_reps_for_levelN(N, 3)
sage: all(len(L[i]) == k.class_number() for i in range(len(L)))
True
```

```
sage: k.<a> = NumberField(x^4 - x^3 -21*x^2 + 17*x + 133)
sage: N = k.ideal(6)
sage: from sage.modular.cusps_nf import NFCusps_ideal_reps_for_levelN
sage: NFCusps_ideal_reps_for_levelN(N)
[(Fractional ideal (1), Fractional ideal (67, a + 17),
  Fractional ideal (127, a + 48),
  Fractional ideal (157, a - 19))]
sage: L = NFCusps_ideal_reps_for_levelN(N, 5)
sage: all(len(L[i]) == k.class_number() for i in range(len(L)))
True
```

`sage.modular.cusps_nf.list_of_representatives()`

Return a list of ideals, coprime to the ideal \( N \), representatives of the ideal classes of the corresponding number field.

Note: This list, used every time we check \( \Gamma_0(N) \) - equivalence of cusps, is cached.

INPUT:
• \( N \) – an ideal of a number field.

**OUTPUT:**

A list of ideals coprime to the ideal \( N \), such that they are representatives of all the ideal classes of the number field.

**EXAMPLES:**

```python
sage: from sage.modular.cusps_nf import list_of_representatives
sage: k.<a> = NumberField(x^4 + 13*x^3 - 11)
  sage: N = k.ideal(713, a + 208)
  sage: L = list_of_representatives(N); L
  (Fractional ideal (1),
   Fractional ideal (47, a - 9),
   Fractional ideal (53, a - 16))
```

**sage.modular.cusps_nf.number_of_Gamma0_NFCusps(\( N \))**

Return the total number of orbits of cusps under the action of the congruence subgroup \( \Gamma_0(N) \).

**INPUT:**

• \( N \) – a number field ideal.

**OUTPUT:**

integer – the number of orbits of cusps under \( \Gamma_0(N) \)-action.

**EXAMPLES:**

```python
sage: k.<a> = NumberField(x^3 + 11)
  sage: N = k.ideal(2, a+1)
  sage: from sage.modular.cusps_nf import number_of_Gamma0_NFCusps
  sage: number_of_Gamma0_NFCusps(N)
  4
  sage: len(L) == number_of_Gamma0_NFCusps(N)
  True
  sage: k.<a> = NumberField(x^2 + 7)
  sage: N = k.ideal(9)
  sage: number_of_Gamma0_NFCusps(N)
  6
  sage: N = k.ideal(a^9 + 7)
  sage: number_of_Gamma0_NFCusps(N)
  24
```

**sage.modular.cusps_nf.units_mod_ideal(I)**

Return integral elements of the number field representing the images of the global units modulo the ideal \( I \).

**INPUT:**

• \( I \) – number field ideal.

**OUTPUT:**

A list of integral elements of the number field representing the images of the global units modulo the ideal \( I \). Elements of the list might be equivalent to each other mod \( I \).

**EXAMPLES:**

```python
```

### 4.15. The set \( \mathbb{P}(K) \) of cusps of a number field \( K \)

437
4.16 Hypergeometric motives

This is largely a port of the corresponding package in Magma. One important conventional difference: the motivic parameter $t$ has been replaced with $1/t$ to match the classical literature on hypergeometric series. (E.g., see [BeukersHeckman])

The computation of Euler factors is currently only supported for primes $p$ of good reduction. That is, it is required that $v_p(t) = v_p(t - 1) = 0$.

AUTHORS:

- Frédéric Chapoton
- Kiran S. Kedlaya

EXAMPLES:

```python
sage: from sage.modular.hypergeometric_motives import HypergeometricData as Hyp
sage: H = Hyp(cyclotomic=([30], [1,2,3,5]))
1.00000000000000
```
True

```sage
H.euler_factor(2, 7)
T^8 + T^5 + T^3 + 1
```

REFERENCES:
- [BeukersHeckman]
- [Benasque2009]
- [Kat1991]
- [MagmaHGM]
- [Fedorov2015]
- [Roberts2017]
- [Roberts2015]
- [BeCoMe]
- [Watkins]

```python
class sage.modular.hypergeometric_motive.HypergeometricData(cyclotomic=None, alpha_beta=None, gamma_list=None):
    Bases: object
    Creation of hypergeometric motives.
    INPUT:
    three possibilities are offered, each describing a quotient of products of cyclotomic polynomials.
    - cyclotomic – a pair of lists of nonnegative integers, each integer \( k \) represents a cyclotomic polynomial \( \Phi_k \)
    - alpha_beta – a pair of lists of rationals, each rational represents a root of unity
    - gamma_list – a pair of lists of nonnegative integers, each integer \( n \) represents a polynomial \( x^n - 1 \)
    In the last case, it is also allowed to send just one list of signed integers where signs indicate to which part the integer belongs to.
    EXAMPLES:
```
sage: from sage.modular.hypergeometric_motive import HypergeometricData as Hyp
sage: Hyp(cyclotomic=([2],[1]))
Hypergeometric data for [1/2] and [0]
sage: Hyp(alpha_beta=([1/2],[0]))
Hypergeometric data for [1/2] and [0]
sage: Hyp(alpha_beta=([1/5,2/5,3/5,4/5],[0,0,0,0]))
Hypergeometric data for [1/5, 2/5, 3/5, 4/5] and [0, 0, 0, 0]
sage: Hyp(gamma_list=(5,)[1,1,1,1]))
Hypergeometric data for [1/5, 2/5, 3/5, 4/5] and [0, 0, 0, 0]
sage: Hyp(gamma_list=(5,)[-1,-1,-1,-1,-1]))
Hypergeometric data for [1/5, 2/5, 3/5, 4/5] and [0, 0, 0, 0]
```
**H_value**(*p, f, t, ring=None*)

Return the trace of the Frobenius, computed in terms of Gauss sums using the hypergeometric trace formula.

**INPUT:**

- *p* – a prime number
- *f* – an integer such that \( q = p^f \)
- *t* – a rational parameter
- *ring* – optional (default UniversalCyclotomicField)

The ring could be also ComplexField(n) or QQbar.

**OUTPUT:**

an integer

**Warning:** This is apparently working correctly as can be tested using ComplexField(70) as value ring. Using instead UniversalCyclotomicField, this is much slower than the \( p \)-adic version \( p\text{adic}_H\text{_value}() \).

**EXAMPLES:**

With values in the UniversalCyclotomicField (slow):

```sage
sage: from sage.modular.hypergeometric_motive import HypergeometricData as Hyp
sage: H = Hyp(alpha_beta=((1/2)*4, [0]*4))

sage: [H.H_value(3,i,-1) for i in range(1,3)]
[0, -12]

sage: [H.H_value(5,i,-1) for i in range(1,3)]
[-4, 276]

sage: [H.H_value(7,i,-1) for i in range(1,3)]  # not tested
[0, -476]

sage: [H.H_value(11,i,-1) for i in range(1,3)]  # not tested
[0, -4972]

sage: [H.H_value(13,i,-1) for i in range(1,3)]  # not tested
[-84, -1420]
```

With values in ComplexField:

```sage
sage: [H.H_value(5,i,-1, ComplexField(60)) for i in range(1,3)]
[-4, 276]
```

Check issue from github issue #28404:

```sage
sage: H1 = Hyp(cyclotomic=[[1,1,1],[6,2]])
sage: H2 = Hyp(cyclotomic=[[6,2],[1,1,1]])

sage: [H1.H_value(5,1,i) for i in range(2,5)]
[1, -4, -4]

sage: [H2.H_value(5,1,QQ(i)) for i in range(2,5)]
[-4, 1, -4]
```

**REFERENCES:**

- [BeCoMe] (Theorem 1.3)
• [Benasque2009]

**M_value()**

Return the $M$ coefficient that appears in the trace formula.

**OUTPUT:**
a rational

**See also:**
canonical_scheme()

**EXAMPLES:**

```python
sage: from sage.modular.hypergeometric_motive import HypergeometricData as Hyp
sage: H = Hyp(alpha_beta=((1/6, 1/3, 2/3, 5/6), (1/8, 3/8, 5/8, 7/8)))

sage: H.M_value()
729/4096

sage: Hyp(alpha_beta=((1/2, 1/2, 1/2, 1/2), (0, 0, 0, 0))).M_value()
256

sage: Hyp(cyclotomic=(5, (1, 1, 1, 1))).M_value()
3125
```

**alpha()**

Return the first tuple of rational arguments.

**EXAMPLES:**

```python
sage: from sage.modular.hypergeometric_motive import HypergeometricData as Hyp

sage: Hyp(alpha_beta=((1/2), (0))).alpha()
[1/2]
```

**alpha_beta()**

Return the pair of lists of rational arguments.

**EXAMPLES:**

```python
sage: from sage.modular.hypergeometric_motive import HypergeometricData as Hyp

sage: Hyp(alpha_beta=((1/2), (0))).alpha_beta()
([1/2], [0])
```

**beta()**

Return the second tuple of rational arguments.

**EXAMPLES:**

```python
sage: from sage.modular.hypergeometric_motive import HypergeometricData as Hyp

sage: Hyp(alpha_beta=((1/2), (0))).beta()
[0]
```

canonical_scheme($t=None$)

Return the canonical scheme.

This is a scheme that contains this hypergeometric motive in its cohomology.

**EXAMPLES:**
sage: from sage.modular.hypergeometric_motive import HypergeometricData as Hyp
sage: H = Hyp(cyclotomic=([3],[4]))
sage: H.gamma_list()
[-1, 2, 3, -4]
sage: H.canonical_scheme()
Spectrum of Quotient of Multivariate Polynomial Ring
in X0, X1, Y0, Y1 over Fraction Field of Univariate Polynomial Ring
in t over Rational Field by the ideal
(X0 + X1 - 1, Y0 + Y1 - 1, (-t)*X0^2*X1^3 + 27/64*Y0*Y1^4)

sage: H = Hyp(gamma_list=[-2, 3, 4, -5])
sage: H.canonical_scheme()
Spectrum of Quotient of Multivariate Polynomial Ring
in X0, X1, Y0, Y1 over Fraction Field of Univariate Polynomial Ring
in t over Rational Field by the ideal
(X0 + X1 - 1, Y0 + Y1 - 1, (-t)*X0^3*X1^4 + 1728/3125*Y0^2*Y1^5)

REFERENCES:

[Kat1991], section 5.4

cyclotomic_data()
Return the pair of tuples of indices of cyclotomic polynomials.

EXAMPLES:

sage: from sage.modular.hypergeometric_motive import HypergeometricData as Hyp
sage: Hyp(alpha_beta=([1/2],[0])).cyclotomic_data()
([2], [1])

defining_polynomials()
Return the pair of products of cyclotomic polynomials.

EXAMPLES:

sage: from sage.modular.hypergeometric_motive import HypergeometricData as Hyp
sage: Hyp(alpha_beta=([1/4,3/4],[0,0])).defining_polynomials()
((x^2 + 1, x^2 - 2*x + 1)

degree()
Return the degree.

This is the sum of the Hodge numbers.

See also:
hodge_numbers()

EXAMPLES:

sage: from sage.modular.hypergeometric_motive import HypergeometricData as Hyp
sage: Hyp(alpha_beta=([1/2],[0])).degree()
1
sage: Hyp(gamma_list=([2,2,4],[8])).degree()
4
sage: Hyp(cyclotomic=([5,6],[1,1,2,2,3])).degree()
euler_factor\((t, p, cache_p=False)\)

Return the Euler factor of the motive \(H_t\) at prime \(p\).

**INPUT:**

- \(t\) – rational number, not 0 or 1
- \(p\) – prime number of good reduction

**OUTPUT:**

a polynomial

See [Benasque2009] for explicit examples of Euler factors.

For odd weight, the sign of the functional equation is +1. For even weight, the sign is computed by a recipe found in 11.1 of [Watkins].

**EXAMPLES:**

```python
sage: from sage.modular.hypergeometric_motive import HypergeometricData as Hyp
sage: H = Hyp(alpha_beta=[[1/2]*4,[0]*4))

sage: H.euler_factor(-1, 5)
15625*T^4 + 500*T^3 - 130*T^2 + 4*T + 1

sage: H = Hyp(gamma_list=[-6,-1,4,3])

sage: H.weight(), H.degree()
(1, 2)

sage: [H.euler_factor(1/t,p) for p in [11,13,17,19,23,29]]
[11*T^2 + 4*T + 1,
13*T^2 + 1,
17*T^2 + 1,
19*T^2 + 1,
23*T^2 + 8*T + 1,
29*T^2 + 2*T + 1]

sage: H = Hyp(cyclotomic=(6,2),[1,1]))

sage: H.weight(), H.degree()
(2, 3)

sage: [H.euler_factor(1/4,p) for p in [5,7,11,13,17,19]]
[125*T^3 + 20*T^2 + 4*T + 1,
343*T^3 - 42*T^2 - 6*T + 1,
-1331*T^3 - 22*T^2 + 2*T + 1,
-2197*T^3 + 156*T^2 + 12*T + 1,
4913*T^3 + 323*T^2 + 19*T + 1,
6859*T^3 - 57*T^2 - 3*T + 1]

sage: H = Hyp(alpha_beta=[[1/12,5/12,7/12,11/12],[0,1/2,1/2,1/2]])

sage: H.weight(), H.degree()
```

(continues on next page)
sage: t = -5
sage: [H.euler_factor(1/t, p) for p in [11, 13, 17, 19, 23, 29]]

[-14641*T^4 - 1210*T^3 + 10*T + 1,
 -28561*T^4 - 2704*T^3 + 16*T + 1,
 -83521*T^4 - 4046*T^3 + 14*T + 1,
 130321*T^4 + 14440*T^3 + 969*T^2 + 40*T + 1,
 279841*T^4 - 25392*T^3 + 1242*T^2 - 48*T + 1,
 707281*T^4 - 7569*T^3 + 696*T^2 - 9*T + 1]

This is an example of higher degree:

sage: H = Hyp(cyclotomic=([11], [7, 12]))
sage: H.euler_factor(2, 13)
371293*T^10 - 85683*T^9 + 26364*T^8 + 1352*T^7 - 65*T^6 + 394*T^5 - 5*T^4 + 8*T^3 + 12*T^2 - 3*T + 1

sage: H.euler_factor(2, 19)  # long time
2476099*T^10 - 651605*T^9 + 233206*T^8 - 77254*T^7 + 20349*T^6 - 4611*T^5 + 1071*T^4 - 214*T^3 + 34*T^2 - 5*T + 1

REFERENCES:

• [Roberts2015]
• [Watkins]

gamma_array()  
Return the dictionary \{v : \gamma_v\} for the expression

\[ \prod_v (T^v - 1)^{\gamma_v} \]

EXAMPLES:

sage: from sage.modular.hypergeometric_motive import HypergeometricData as Hyp
sage: Hyp(alpha_beta=([1/2],[0])).gamma_array()
{1: -2, 2: 1}
sage: Hyp(cyclotomic=([6,2],[1,1,1])).gamma_array()
{1: -3, 3: -1, 6: 1}

gamma_list()  
Return a list of integers describing the \(x^n - 1\) factors.

Each integer \(n\) stands for \((x^{|n|} - 1)^{sgn(n)}\).

EXAMPLES:

sage: from sage.modular.hypergeometric_motive import HypergeometricData as Hyp
sage: Hyp(alpha_beta=([1/2],[0])).gamma_list()
[-1, -1, 2]
sage: Hyp(cyclotomic=([6,2],[1,1,1])).gamma_list()
[-1, -1, -1, -3, 6]
sage: Hyp(cyclotomic=([3],[4])).gamma_list()
[-1, 2, 3, -4]
\textbf{gauss\_table}(p, f, prec)

Return (and cache) a table of Gauss sums used in the trace formula.

See also:

\textit{gauss\_table\_full()}

\textbf{EXAMPLES:}

\begin{verbatim}
sage: from sage.modular.hypergeometric_motive import HypergeometricData as Hyp
sage: H = Hyp(cyclotomic=([3], [4]))
sage: H.gauss_table(2, 2, 4)
(4, [1 + 2 + 2^2 + 2^3, 1 + 2 + 2^2 + 2^3, 1 + 2 + 2^2 + 2^3, 1 + 2 + 2^2 + 2^3])
\end{verbatim}

\textbf{gauss\_table\_full()}

Return a dict of all stored tables of Gauss sums.

The result is passed by reference, and is an attribute of the class; consequently, modifying the result has
global side effects. Use with caution.

See also:

\textit{gauss\_table()}

\textbf{EXAMPLES:}

\begin{verbatim}
sage: from sage.modular.hypergeometric_motive import HypergeometricData as Hyp
sage: H = Hyp(cyclotomic=([3], [4]))
sage: H.euler_factor(2, 7, cache_p=True)
7*T^2 - 3*T + 1
sage: H.gauss_table_full()[(7, 1)]
(2, array(['1', [-1, -29, -25, -48, -47, -22]))
\end{verbatim}

Clearing cached values:

\begin{verbatim}
sage: H = Hyp(cyclotomic=([3], [4]))
sage: H.euler_factor(2, 7, cache_p=True)
7*T^2 - 3*T + 1
sage: d = H.gauss_table_full()
sage: d.clear() # Delete all entries of this dict
sage: H1 = Hyp(cyclotomic=([5], [12]))
sage: d1 = H1.gauss_table_full()
sage: len(d1.keys()) # No cached values
0
\end{verbatim}

\textbf{has\_symmetry\_at\_one()}

If True, the motive H(t=1) is a direct sum of two motives.

Note that simultaneous exchange of (t,1/t) and (alpha,beta) always gives the same motive.

\textbf{EXAMPLES:}

\begin{verbatim}
sage: from sage.modular.hypergeometric_motive import HypergeometricData as Hyp
sage: Hyp(alpha_beta=[[1/2]*16, [0]*16]).has_symmetry_at_one() True
\end{verbatim}

\textbf{REFERENCES:}

- [Roberts2017]
**hodge_function(x)**

Evaluate the Hodge polygon as a function.

See also:

- **hodge_numbers()**, **hodge_polynomial()**, **hodge_polygon_vertices()**

**EXAMPLES:**

```python
sage: from sage.modular.hypergeometric_motive import HypergeometricData as Hyp
sage: H = Hyp(cyclotomic=([6,10],[3,12]))
sage: H.hodge_function(3)
2
sage: H.hodge_function(4)
4
```

**hodge_numbers()**

Return the Hodge numbers.

See also:

- **degree()**, **hodge_polynomial()**, **hodge_polygon()**

**EXAMPLES:**

```python
sage: from sage.modular.hypergeometric_motive import HypergeometricData as Hyp
sage: H = Hyp(cyclotomic=([3],[6]))
sage: H.hodge_numbers()
[1, 1]
sage: H = Hyp(cyclotomic=([4],[1,2]))
sage: H.hodge_numbers()
[2]
sage: H = Hyp(gamma_list=([8,2,2,2],[6,4,3,1]))
sage: H.hodge_numbers()
[1, 2, 2, 1]
sage: H = Hyp(gamma_list=([5],[1,1,1,1,1]))
sage: H.hodge_numbers()
[1, 1, 1, 1]
sage: H = Hyp(gamma_list=[6,1,-4,-3])
sage: H.hodge_numbers()
[1, 1]
sage: H = Hyp(gamma_list=[-3]*4 + [1]*12)
sage: H.hodge_numbers()
[1, 1, 1, 1, 1, 1]
```

**REFERENCES:**

- [Fedorov2015]

**hodge_polygon_vertices()**

Return the vertices of the Hodge polygon.
See also:

\texttt{hodge_numbers()}, \texttt{hodge_polynomial()}, \texttt{hodge_function()}

EXAMPLES:

```python
sage: from sage.modular.hypergeometric_motive import HypergeometricData as Hyp
sage: H = Hyp(cyclotomic=([5,10],[3,12]))
```

```diff
    sage: H.hodge_polygon_vertices()
    [(0, 0), (1, 0), (3, 2), (5, 6), (6, 9)]
```

```python
sage: H = Hyp(cyclotomic=([2,2,2,2,3,3,3,6,6],[1,1,4,5,9]))
```

```diff
    sage: H.hodge_polygon_vertices()
    [(0, 0), (1, 0), (4, 3), (7, 9), (10, 18), (13, 30), (14, 35)]
```

\textbf{hodge_polynomial()}

Return the Hodge polynomial.

See also:

\texttt{hodge_numbers()}, \texttt{hodge_polygon_vertices()}, \texttt{hodge_function()}

EXAMPLES:

```python
sage: from sage.modular.hypergeometric_motive import HypergeometricData as Hyp
sage: H = Hyp(cyclotomic=([6,10],[3,12]))
```

```diff
    sage: H.hodge_polynomial()
    (T^3 + 2*T^2 + 2*T + 1)/T^2
```

```python
sage: H = Hyp(cyclotomic=([2,2,2,2,3,3,3,6,6],[1,1,4,5,9]))
```

```diff
    sage: H.hodge_polynomial()
    (T^5 + 3*T^4 + 3*T^3 + 3*T^2 + 3*T + 1)/T^2
```

\textbf{is_primitive()}

Return whether this data is primitive.

See also:

\texttt{primitive_index()}, \texttt{primitive_data()}

EXAMPLES:

```python
sage: from sage.modular.hypergeometric_motive import HypergeometricData as Hyp
sage: Hyp(cyclotomic=([3],[4])).is_primitive()
```

```diff
    True
```

```python
sage: Hyp(gamma_list=[-2, 4, 6, -8]).is_primitive()
```

```diff
    False
```

```python
sage: Hyp(gamma_list=[-3, 6, 9, -12]).is_primitive()
```

```diff
    False
```

\textbf{padic_H_value}(p,f,t,prec=None, cache_p=False)

Return the \(p\)-adic trace of Frobenius, computed using the Gross-Koblitz formula.

If left unspecified, \(\texttt{prec}\) is set to the minimum \(p\)-adic precision needed to recover the Euler factor.

If \(\texttt{cache_p}\) is True, then the function caches an intermediate result which depends only on \(p\) and \(f\). This leads to a significant speedup when iterating over \(t\).

INPUT:

- \(p\) – a prime number

4.16. Hypergeometric motives

447
• $f$ – an integer such that $q = p^f$
• $t$ – a rational parameter
• prec – precision (optional)
• cache_p – a boolean

OUTPUT:
an integer

EXAMPLES:
From Benasque report [Benasque2009], page 8:

```
sage: from sage.modular.hypergeometric_motive import HypergeometricData as Hyp
sage: H = Hyp(alpha_beta=(\[1/2\]^4,\[0\]^4))
```
```
sage: [H.padic_H_value(3,i,-1) for i in range(1,3)]
[0, -12]
sage: [H.padic_H_value(5,i,-1) for i in range(1,3)]
[-4, 276]
sage: [H.padic_H_value(7,i,-1) for i in range(1,3)]
[0, -476]
sage: [H.padic_H_value(11,i,-1) for i in range(1,3)]
[0, -4972]
```

From [Roberts2015] (but note conventions regarding $t$):

```
sage: H = Hyp(gamma_list=[-6,-1,4,3])
sage: t = 189/125
sage: H.padic_H_value(13,1,1/t)
0
```

REFERENCES:
• [MagmaHGM]

**primitive_data()**
Return a primitive version.

See also:

*is_primitive(), primitive_index()*

EXAMPLES:

```
sage: from sage.modular.hypergeometric_motive import HypergeometricData as Hyp
sage: H = Hyp(cyclotomic=(\[3\],\[4\]))
```
```
sage: H2 = Hyp(gamma_list=[-2, 4, 6, -8])
sage: H2.primitive_data() == H
True
```

**primitive_index()**
Return the primitive index.

See also:

*is_primitive(), primitive_data()*

EXAMPLES:
sage: from sage.modular.hypergeometric_motive import HypergeometricData as Hyp
sage: Hyp(cyclotomic=(3,4)).primitive_index()
1
sage: Hyp(gamma_list=[-2, 4, 6, -8]).primitive_index()
2
sage: Hyp(gamma_list=[-3, 6, 9, -12]).primitive_index()
3

\textbf{sign}(t, p)

Return the sign of the functional equation for the Euler factor of the motive $H_t$ at the prime $p$.

For odd weight, the sign of the functional equation is $+1$. For even weight, the sign is computed by a recipe found in 11.1 of [Watkins] (when 0 is not in alpha).

**EXAMPLES:**

sage: from sage.modular.hypergeometric_motive import HypergeometricData as Hyp
sage: H = Hyp(cyclotomic=(6,2), [1,1,1])
sage: H.weight(), H.degree()
(2, 3)
sage: [H.sign(1/4,p) for p in [5,7,11,13,17,19]]
[1, 1, -1, -1, 1, 1]
sage: H = Hyp(alpha_beta=(1/12,5/12,7/12,11/12), [0,1/2,1/2,1/2])
sage: H.weight(), H.degree()
(2, 4)
sage: t = -5
sage: [H.sign(1/t,p) for p in [11,13,17,19,23,29]]
[-1, -1, -1, 1, 1, 1]

We check that github issue #28404 is fixed:

sage: H = Hyp(cyclotomic=(1,1,1), [6,2])
sage: [H.sign(4,p) for p in [5,7,11,13,17,19]]
[1, 1, -1, -1, 1, 1]

\textbf{swap_alpha_beta}()

Return the hypergeometric data with alpha and beta exchanged.

**EXAMPLES:**

sage: from sage.modular.hypergeometric_motive import HypergeometricData as Hyp
sage: H = Hyp(alpha_beta=(1/2), [0])
sage: H.swap_alpha_beta()
Hypergeometric data for [0] and [1/2]

\textbf{trace}(p, f, t, prec=None, cache_p=False)

Return the $p$-adic trace of Frobenius, computed using the Gross-Koblitz formula.

If left unspecified, \texttt{prec} is set to the minimum $p$-adic precision needed to recover the Euler factor.

If \texttt{cache}_p is True, then the function caches an intermediate result which depends only on \texttt{p} and \texttt{f}. This leads to a significant speedup when iterating over \texttt{t}.

**INPUT:**

- \texttt{p} – a prime number
• $f$ – an integer such that $q = p^f$
• $t$ – a rational parameter
• prec – precision (optional)
• cache_p - a boolean

OUTPUT:

an integer

EXAMPLES:

From Benasque report [Benasque2009], page 8:

```
sage: from sage.modular.hypergeometric_motive import HypergeometricData as Hyp
sage: H = Hyp(alpha_beta=(1/2, 0))
```
```
sage: [H.padic_H_value(3, i, -1) for i in range(1, 3)]
[0, -12]
```
```
sage: [H.padic_H_value(5, i, -1) for i in range(1, 3)]
[-4, 276]
```
```
sage: [H.padic_H_value(7, i, -1) for i in range(1, 3)]
[0, -476]
```
```
sage: [H.padic_H_value(11, i, -1) for i in range(1, 3)]
[0, -4972]
```

From [Roberts2015] (but note conventions regarding $t$):

```
sage: H = Hyp(gamma_list=[-6, -1, 4, 3])
sage: t = 189/125
sage: H.padic_H_value(13, 1, 1/t)
0
```

REFERENCES:

• [MagmaHGM]

twist()

Return the twist of this data.

This is defined by adding $1/2$ to each rational in $\alpha$ and $\beta$.

This is an involution.

EXAMPLES:

```
sage: from sage.modular.hypergeometric_motive import HypergeometricData as Hyp
sage: H = Hyp(alpha_beta=(1/2, 0))
sage: H.twist()
Hypergeometric data for [0] and [1/2]
sage: H.twist().twist() == H
True
```

weight()

Return the motivic weight of this motivic data.

EXAMPLES:
With rational inputs:

```
sage: from sage.modular.hypergeometric_motive import HypergeometricData as Hyp
sage: Hyp(alpha_beta=(1/2,0)).weight()
0
sage: Hyp(alpha_beta=(1/4,3/4,0,0)).weight()
1
sage: Hyp(alpha_beta=(1/6,1/3,2/3,5/6,0,0,1/4,3/4)).weight()
1
sage: H = Hyp(alpha_beta=(1/6,1/3,2/3,5/6,1/8,3/8,5/8,7/8))
sage: H.weight()
1
```

With cyclotomic inputs:

```
sage: Hyp(cyclotomic=(6,2,1,1)).weight()
2
sage: Hyp(cyclotomic=(6,2)).weight()
0
sage: Hyp(cyclotomic=(8,2,3)).weight()
0
sage: Hyp(cyclotomic=(5,1,1,1)).weight()
3
sage: Hyp(cyclotomic=(5,6,1,2,2,3)).weight()
1
sage: Hyp(cyclotomic=(3,8,1,1,2,6)).weight()
2
sage: Hyp(cyclotomic=(3,2,2,4)).weight()
1
```

With gamma list input:

```
sage: Hyp(gamma_list=(8,2,2,2,6,4,3,1)).weight()
3
```

```
wild_primes()
Return the wild primes.
EXAMPLES:
```
```
sage: from sage.modular.hypergeometric_motive import HypergeometricData as Hyp
sage: Hyp(cyclotomic=(3,4)).wild_primes()
[2, 3]
sage: Hyp(cyclotomic=(2,2,2,3,3,3,3,6,6,1,1,4,5,9)).wild_primes()
[2, 3, 5]
```

```
zigzag(x, flip_beta=False)
Count alpha’s at most x minus beta’s at most x.
This function is used to compute the weight and the Hodge numbers. With \( \text{flip}_\beta \) set to True, replace each \( b \) in \( \beta \) with \( 1 - b \).
See also:

weight(), hodge_numbers()
```

EXAMPLES: 4.16. Hypergeometric motives 451
sage: from sage.modular.hypergeometric_motive import HypergeometricData as Hyp
sage: H = Hyp(alpha_beta=[[1/6,1/3,2/3,5/6],[1/8,3/8,5/8,7/8]])
sage: [H.zigzag(x) for x in [0, 1/3, 1/2]]
[0, 1, 0]
sage: H = Hyp(cyclotomic=[[5],[1,1,1,1]])
[4, 4, 3, 2, 1, -1, -2, -3, -4]
sage: [H.zigzag(x) for x in [0,1/6,1/4,1/2,3/4,5/6]]
[4, 4, 3, 2, 1, -1, -2, -3, -4]

sage.modular.hypergeometric_motive.alpha_to_cyclotomic(alpha)
Convert from a list of rationals arguments to a list of integers.
The input represents arguments of some roots of unity.
The output represents a product of cyclotomic polynomials with exactly the given roots. Note that the multiplicity of $r/s$ in the list must be independent of $r$; otherwise, a ValueError will be raised.
This is the inverse of cyclotomic_to_alpha().

EXAMPLES:

sage: from sage.modular.hypergeometric_motive import alpha_to_cyclotomic
sage: alpha_to_cyclotomic([0])
[1]
sage: alpha_to_cyclotomic([1/2])
[2]
sage: alpha_to_cyclotomic([1/5,2/5,3/5,4/5])
[5]
sage: alpha_to_cyclotomic([0, 1/6, 1/3, 1/2, 2/3, 5/6])
[1, 2, 3, 6]
sage: alpha_to_cyclotomic([1/3,2/3,1/2])
[2, 3]

sage.modular.hypergeometric_motive.capital_M(n)
Auxiliary function, used to describe the canonical scheme.

INPUT:
- n – an integer

OUTPUT:
a rational

EXAMPLES:

sage: from sage.modular.hypergeometric_motive import capital_M
sage: [capital_M(i) for i in range(1,8)]
[1, 4, 27, 64, 3125, 432, 823543]

sage.modular.hypergeometric_motive.characteristic_polynomial_from_traces(traces, d, q, i, sign)
Given a sequence of traces $t_1, \ldots, t_k$, return the corresponding characteristic polynomial with Weil numbers as roots.
The characteristic polynomial is defined by the generating series

$$P(T) = \exp \left( - \sum_{k \geq 1} t_k \frac{T^k}{k} \right)$$
and should have the property that reciprocals of all roots have absolute value $q^{i/2}$.

**INPUT:**
- **traces** – a list of integers $t_1, \ldots, t_k$
- **d** – the degree of the characteristic polynomial
- **q** – power of a prime number
- **i** – integer, the weight in the motivic sense
- **sign** – integer, the sign

**OUTPUT:**

a polynomial

**EXAMPLES:**

```python
sage: from sage.modular.hypergeometric_motive import characteristic_polynomial_from_traces
sage: characteristic_polynomial_from_traces([1, 1], 1, 3, 0, -1)
-T + 1
sage: characteristic_polynomial_from_traces([25], 1, 5, 4, -1)
-25*T + 1
sage: characteristic_polynomial_from_traces([3], 2, 5, 1, 1)
5*T^2 - 3*T + 1
sage: characteristic_polynomial_from_traces([1], 2, 7, 1, 1)
7*T^2 - T + 1
sage: characteristic_polynomial_from_traces([20], 3, 29, 2, 1)
24389*T^3 - 580*T^2 - 20*T + 1
sage: characteristic_polynomial_from_traces([12], 3, 13, 2, -1)
-2197*T^3 + 156*T^2 - 12*T + 1
```

**sage.modular.hypergeometric_motive.cyclotomic_to_alpha**(cyclo)

Convert a list of indices of cyclotomic polynomials to a list of rational numbers.

The input represents a product of cyclotomic polynomials.

The output is the list of arguments of the roots of the given product of cyclotomic polynomials.

This is the inverse of **alpha_to_cyclotomic()**.

**EXAMPLES:**

```python
sage: from sage.modular.hypergeometric_motive import cyclotomic_to_alpha
sage: cyclotomic_to_alpha([1])
[0]
```
sage: cyclotomic_to_alpha([2])
[1/2]
sage: cyclotomic_to_alpha([5])
[1/5, 2/5, 3/5, 4/5]
sage: cyclotomic_to_alpha([1,2,3,6])
[0, 1/6, 1/3, 1/2, 2/3, 5/6]
sage: cyclotomic_to_alpha([2,3])
[1/3, 1/2, 2/3]

\texttt{sage.modular.hypergeometric_motive.cyclotomic\_to\_gamma(cyclo\_up, cyclo\_down)}

Convert a quotient of products of cyclotomic polynomials to a quotient of products of polynomials $x^n - 1$.

INPUT:

• cyclo\_up – list of indices of cyclotomic polynomials in the numerator
• cyclo\_down – list of indices of cyclotomic polynomials in the denominator

OUTPUT:

a dictionary mapping an integer \(n\) to the power of \(x^n - 1\) that appears in the given product

EXAMPLES:

\begin{verbatim}
sage: from sage.modular.hypergeometric_motive import cyclotomic_to_gamma
sage: cyclotomic_to_gamma([6], [1])
{2: -1, 3: -1, 6: 1}
\end{verbatim}

\texttt{sage.modular.hypergeometric_motive.enumerate\_hypergeometric\_data(d, weight=None)}

Return an iterator over parameters of hypergeometric motives (up to swapping).

INPUT:

• \(d\) – the degree
• \texttt{weight} – optional integer, to specify the motivic weight

EXAMPLES:

\begin{verbatim}
sage: from sage.modular.hypergeometric_motive import enumerate_hypergeometric_data
˓→
sage: l = [H for H in enum(6, weight=2) if H.hodge_numbers()[0] == 1]
sage: len(l)
112
\end{verbatim}

\texttt{sage.modular.hypergeometric_motive.gamma\_list\_to\_cyclotomic(galist)}

Convert a quotient of products of polynomials $x^n - 1$ to a quotient of products of cyclotomic polynomials.

INPUT:

• galist – a list of integers, where an integer \(n\) represents the power $(x^n - 1)^{\text{sgn}(n)}$

OUTPUT:

a pair of list of integers, where \(k\) represents the cyclotomic polynomial $\Phi_k$

EXAMPLES:
```python
sage: from sage.modular.hypergeometric_motive import gamma_list_to_cyclotomic
sage: gamma_list_to_cyclotomic([-1, -1, 2])
([2], [1])
sage: gamma_list_to_cyclotomic([-1, -1, -1, -3, 6])
([2, 6], [1, 1, 1])
sage: gamma_list_to_cyclotomic([-1, 2, 3, -4])
([3], [4])
sage: gamma_list_to_cyclotomic([8,2,2,2,-6,-4,-3,-1])
([2, 2, 8], [3, 3, 6])
```

```
sage.modular.hypergeometric_motive.possible_hypergeometric_data(d, weight=None)

Return the list of possible parameters of hypergeometric motives (up to swapping).

INPUT:

• d – the degree
• weight – optional integer, to specify the motivic weight

EXAMPLES:
```python
sage: from sage.modular.hypergeometric_motive import possible_hypergeometric_data
    as P
sage: [len(P(i,weight=2)) for i in range(1, 7)]
[0, 0, 10, 30, 93, 234]
```

### 4.17 Algebra of motivic multiple zeta values

This file contains an implementation of the algebra of motivic multiple zeta values.

The elements of this algebra are not the usual multiple zeta values as real numbers defined by concrete iterated integrals, but abstract symbols that satisfy all the linear relations between formal iterated integrals that come from algebraic geometry (motivic relations). Although this set of relations is not explicit, one can test the equality as explained in the article [Brown2012]. One can map these motivic multiple zeta values to the associated real numbers. Conjecturally, this period map should be injective.

The implementation follows closely all the conventions from [Brown2012].

As a convenient abbreviation, the elements will be called multizetas.

EXAMPLES:

One can input multizetas using compositions as arguments:

```python
sage: Multizeta(3)
ζ(3)
sage: Multizeta(2,3,2)
ζ(2,3,2)
```

as well as linear combinations of them:

```python
sage: Multizeta(5)+6*Multizeta(2,3)
6*ζ(2,3) + ζ(5)
```
This creates elements of the class `Multizetas`.

One can multiply such elements:

```sage
Multizeta(2)*Multizeta(3)
6*ζ(1,4) + 3*ζ(2,3) + ζ(3,2)
```

and their linear combinations:

```sage
Multizeta(2)+Multizeta(1,2)*Multizeta(3)
9*ζ(1,1,4) + 5*ζ(1,2,3) + 2*ζ(1,3,2) + 6*ζ(1,4) + 2*ζ(2,1,3) + ζ(2,2,2) + 3*ζ(2,3) + ζ(3,1,2) + ζ(3,2)
```

The algebra is graded by the weight, which is the sum of the arguments. One can extract homogeneous components:

```sage
z = Multizeta(6)+6*Multizeta(2,3)
z.homogeneous_component(5)
6*ζ(2,3)
```

One can also use the ring of multiple zeta values as a base ring for other constructions:

```sage
Z = Multizeta
M = matrix(2,2,[Z(2),Z(3),Z(4),Z(5)])
M.det()
-10*ζ(1,6) - 5*ζ(2,5) - ζ(3,4) + ζ(4,3) + ζ(5,2)
```

### Auxiliary class for alternative notation

One can also use sequences of 0 and 1 as arguments:

```sage
Multizeta(1,1,0)+3*Multizeta(1,0,0)
I(110) + 3*I(100)
```

This creates an element of the auxiliary class `Multizetas_iterated`. This class is used to represent multiple zeta values as iterated integrals.

One can also multiply such elements:

```sage
Multizeta(1,0)*Multizeta(1,0)
4*I(1100) + 2*I(1010)
```

Back-and-forth conversion between the two classes can be done using the methods “composition” and “iterated”:

```sage
(Multizeta(2)*Multizeta(3)).iterated()
6*I(1100) + 3*I(10100) + I(10010)

(Multizeta(1,0)*Multizeta(1,0)).composition()
4*ζ(1,3) + 2*ζ(2,2)
```

Beware that the conversion between these two classes, besides exchanging the indexing by words in 0 and 1 and the indexing by compositions, also involves the sign \((-1)^w\) where \(w\) is the length of the composition and the number of 1 in the associated word in 0 and 1. For example, one has the equality

\[
ζ(2,3,4) = (-1)^3I(1,0,1,0,0,1,0,0,0,0).
\]
Approximate period map

The period map, or rather an approximation, is also available under the generic numerical approximation method:

```
sage: z = Multizeta(5)+6*Multizeta(2,3)
sage: z.n()
2.40979014076349
sage: z.n(prec=100)
2.4097901407634924849438423801
```

Behind the scene, all the numerical work is done by the PARI implementation of numerical multiple zeta values.

Searching for linear relations

All this can be used to find linear dependencies between any set of multiple zeta values. Let us illustrate this by an example.

Let us first build our sample set:

```
sage: Z = Multizeta
sage: L = [Z(*c) for c in [(1, 1, 4), (1, 2, 3), (1, 5), (6,)]]
```

Then one can compute the space of relations:

```
sage: M = matrix([Zc.phi_as_vector() for Zc in L])
sage: K = M.kernel(); K
Vector space of degree 4 and dimension 2 over Rational Field
Basis matrix:
[ 1 0 -2 1/16]
[ 0 1 6 -13/48]
```

and check that the first relation holds:

```
ζ(1,1,4) - 2*ζ(1,5) + 1/16*ζ(6)
sage: relation.phi()
0
sage: relation.is_zero()
True
```

**Warning:** Because this code uses an hardcoded multiplicative basis that is available up to weight 17 included, some parts will not work in larger weights, in particular the test of equality.

REFERENCES:

```python
class sage.modular.multiple_zeta.All_iterated(R)
Bases: CombinatorialFreeModule

Auxiliary class for multiple zeta value as generalized iterated integrals.

This is used to represent multiple zeta values as possibly divergent iterated integrals of the differential forms
ω_0 = dt/t and ω_1 = dt/(t - 1).

This means that the elements are symbols I(a_0; a_1, a_2, ...a_n; a_{n+1}) where all arguments, including the starting
and ending points can be 0 or 1.
```

4.17. Algebra of motivic multiple zeta values 457
This comes with a “regularise” method mapping to \texttt{Multizetas\_iterated}.

**EXAMPLES:**

```python
sage: from sage.modular.multiple_zeta import All\_iterated
sage: M = All\_iterated(QQ); M
Space of motivic multiple zeta values as general iterated integrals over Rational Field
sage: M((0,1,0,1))
I(0;10;1)
sage: x = M((1,1,0,0)); x
I(1;10;0)
sage: x.regularise()
-I(10)
```

**class Element**

**Bases:** IndexedFreeModuleElement

**conversion()**

Conversion to the \texttt{Multizetas\_iterated}.

This assumed that the element has been prepared.

Not to be used directly.

**EXAMPLES:**

```python
sage: from sage.modular.multiple_zeta import All\_iterated
sage: M = All\_iterated(QQ)
sage: x = Word((0,1,0,0,1))
sage: y = M(x).conversion(); y
I(100)
sage: y.parent()
Algebra of motivic multiple zeta values as convergent iterated integrals over Rational Field
```

**regularise()**

Conversion to the \texttt{Multizetas\_iterated}.

This is the regularisation procedure, done in several steps.

**EXAMPLES:**

```python
sage: from sage.modular.multiple_zeta import All\_iterated
sage: M = All\_iterated(QQ)
sage: x = Word((0,0,1,0,1))
M(x).regularise()
-2*I(100)
sage: x = Word((0,1,1,0,1))
M(x).regularise()
I(110)
sage: x = Word((1,0,1,0,0))
M(x).regularise()
2*I(100)
```
**dual()**

Reverse words and exchange the letters 0 and 1.

This is the operation R4 in [Brown2012].

This should be used only when \(a_0 = 0\) and \(a_{n+1} = 1\).

**EXAMPLES:**

```
sage: from sage.modular.multiple_zeta import All_iterated
sage: M = All_iterated(QQ)
sage: x = Word((0,0,1,1,1))
sage: y = Word((0,0,1,0,1))
sage: M.dual(M(x)+5*M(y))
5*I(0;010;1) - I(0;001;1)
```

**dual_on_basis(w)**

Reverse the word and exchange the letters 0 and 1.

This is the operation R4 in [Brown2012].

This should be used only when \(a_0 = 0\) and \(a_{n+1} = 1\).

**EXAMPLES:**

```
sage: from sage.modular.multiple_zeta import All_iterated
sage: M = All_iterated(QQ)
sage: x = Word((0,0,1,0,1))
sage: M.dual_on_basis(x)
I(0;010;1)
sage: x = Word((0,1,0,1,1))
sage: M.dual_on_basis(x)
-I(0;010;1)
```

**expand()**

Perform an expansion as a linear combination.

This is the operation R2 in [Brown2012].

This should be used only when \(a_0 = 0\) and \(a_{n+1} = 1\).

**EXAMPLES:**

```
sage: from sage.modular.multiple_zeta import All_iterated
sage: M = All_iterated(QQ)
sage: x = Word((0,0,1,0,1))
sage: y = Word((0,0,1,1,1))
sage: M.expand(M(x)+2*M(y))
-2*I(0;110;1) - 2*I(0;101;1) - 2*I(0;100;1)
sage: M.expand(M([0,1,1,0,1]))
I(0;110;1)
sage: M.expand(M([0,1,0,0,1]))
I(0;100;1)
```

**expand_on_basis(w)**

Perform an expansion as a linear combination.

This is the operation R2 in [Brown2012].
This should be used only when $a_0 = 0$ and $a_{n+1} = 1$.

EXAMPLES:

```python
sage: from sage.modular.multiple_zeta import All_iterated
sage: M = All_iterated(QQ)
sage: x = Word((0,0,1,0,1))
sage: M.expand_on_basis(x)
-2*I(0;100;1)
```

```python
sage: x = Word((0,0,0,1,0,1,0,0,1))
sage: M.expand_on_basis(x)
6*I(0;1010000;1) + 6*I(0;1001000;1) + 3*I(0;1000100;1)
```

```python
sage: x = Word((0,1,1,0,1))
sage: M.expand_on_basis(x)
I(0;110;1)
```

`reversal()`
Reverse words if necessary.
This is the operation R3 in [Brown2012].
This reverses the word only if $a_0 = 0$ and $a_{n+1} = 1$.

EXAMPLES:

```python
sage: from sage.modular.multiple_zeta import All_iterated
sage: M = All_iterated(QQ)
sage: x = Word((1,0,1,0,0))
sage: y = Word((0,0,1,1,1))
sage: M.reversal(M(x)+2*M(y))
2*I(0;011;1) - I(0;010;1)
```

`reversal_on_basis(w)`
Reverse the word if necessary.
This is the operation R3 in [Brown2012].
This reverses the word only if $a_0 = 0$ and $a_{n+1} = 1$.

EXAMPLES:

```python
sage: from sage.modular.multiple_zeta import All_iterated
sage: M = All_iterated(QQ)
sage: x = Word((1,0,1,0,0))
sage: M.reversal_on_basis(x)
-I(0;010;1)
```

```python
sage: x = Word((0,0,1,1,1))
sage: M.reversal_on_basis(x)
I(0;011;1)
```

`sage.modular.multiple_zeta.D_on_compo(k, compo)`
Return the value of the operator $D_k$ on a multiple zeta value.
This is now only used as a place to keep many doctests.

INPUT:
• $k$ – an odd integer
• compo – a composition

EXAMPLES:

```python
sage: from sage.modular.multiple_zeta import D_on_compo
sage: D_on_compo(3,(2,3))
3*I(100) # I(10)
sage: D_on_compo(3,(4,3))
I(100) # I(1000)
sage: D_on_compo(5,(4,3))
10*I(10000) # I(10)
sage: [D_on_compo(k, [3,5]) for k in (3,5,7)]
[0, -5*I(10000) # I(100), 0]
sage: [D_on_compo(k, [3,7]) for k in (3,5,7,9)]
[0, -6*I(10000) # I(10000), -14*I(1000000) # I(100), 0]
sage: D_on_compo(3,(4,3,3))
-I(100) # I(1000100)
sage: D_on_compo(5,(4,3,3))
-10*I(10000) # I(10100)
sage: D_on_compo(7,(4,3,3))
4*I(1001000) # I(100) + 2*I(1000100) # I(100)
sage: [D_on_compo(k,(1,3,1,3,1,3)) for k in range(3,10,2)]
[0, 0, 0, 0]
```

`sage.modular.multiple_zeta.Multizeta(*args)`

Common entry point for multiple zeta values.

If the argument is a sequence of 0 and 1, an element of `Multizetas_iterated` will be returned.

Otherwise, an element of `Multizetas` will be returned.

The base ring is $\mathbb{Q}$.

EXAMPLES:

```python
sage: Z = Multizeta
sage: Z(1,0,1,0)
I(10)
sage: Z(3,2,2)
ζ(3,2,2)
```

`class sage.modular.multiple_zeta.MultizetaValues`

Bases: `UniqueRepresentation`

Custom cache for numerical values of multiple zetas.

Computations are performed using the PARI/GP `pari:zetamultall` (for the cache) and `pari:zetamult` (for indices/precision outside of the cache).

EXAMPLES:
```
sage: from sage.modular.multiple_zeta import MultizetaValues
sage: M = MultizetaValues()

sage: M((1,2))
1.202056903159594285399738161511449990764986292340...

sage: parent(M((2,3)))
Real Field with 1024 bits of precision

sage: M((2,3), prec=53)
0.228810397603354

sage: parent(M((2,3), prec=53))
Real Field with 53 bits of precision

sage: M((2,3), reverse=False) == M((3,2))
True

sage: M((2,3,4,5))
2.9182061974731261426525583710934944310404272413...e-6

sage: M((2,3,4,5), reverse=False)
0.0011829360522243605614404196778185433287651...

sage: parent(M((2,3,4,5)))
Real Field with 1024 bits of precision

sage: parent(M((2,3,4,5), prec=128))
Real Field with 128 bits of precision

pari_eval(index)

reset(max_weight=8, prec=1024)
    Reset the cache to its default values or to given arguments.

update(max_weight, prec)
    Compute and store more values if needed.

class sage.modular.multiple_zeta.Multizetas(R)

    Bases: CombinatorialFreeModule

    Main class for the algebra of multiple zeta values.

    The convention is chosen so that \(\zeta(1, 2)\) is convergent.

    EXAMPLES:

    sage: M = Multizetas(QQ)
sage: x = M((2,))

sage: y = M((4,3))

sage: x+5*y
\(\zeta(2) + 5\zeta(4,3)\)

sage: x^5*y
\(6^5\zeta(1,4,4) + 8^5\zeta(1,5,3) + 3^5\zeta(2,3,4) + 4^5\zeta(2,4,3) + 3^5\zeta(3,2,4)\)

sage: x^3*y
\(2^3\zeta(3,3,3) + 6^3\zeta(4,1,4) + 3^3\zeta(4,2,3) + \zeta(4,3,2)\)

class Element

    Bases: IndexedListModuleElement
```
is_zero()

Return whether this element is zero.

EXAMPLES:

```sage
def is_zero():
    return False
```

iterated()

Convert to the algebra of iterated integrals.

Beware that this conversion involves signs.

EXAMPLES:

```sage
def iterated():
    return True
```

numerical_approx(prec=None, digits=None, algorithm=None)

Return a numerical value for this element.

EXAMPLES:

```sage
def numerical_approx():
    return 1.202056903
```

If you plan to use intensively numerical approximation at high precision, you might want to add more values and/or accuracy to the cache:
phi()

Return the image of self by the morphism phi.

This sends multiple zeta values to the auxiliary F-algebra.

EXAMPLES:

```sage
M = Multizetas(QQ)
M((1,2)).phi()
f3
```

phi_as_vector()

Return the image of self by the morphism phi as a vector.

The morphism phi sends multiple zeta values to the algebra F_ring(). Then the image is expressed as a vector in a fixed basis of one graded component of this algebra.

This is only defined for homogeneous elements.

EXAMPLES:

```sage
M = Multizetas(QQ)
M((3,2)).phi_as_vector()
(9/2, -2)
M(0).phi_as_vector()
()```

simplify()

Gather terms using the duality relations.

This can help to lower the number of monomials.

EXAMPLES:

```sage
M = Multizetas(QQ)
z = 3*zeta(3,3) + 5*zeta(1,2)
z.simplify()
8*zeta(3)
```

simplify_full(basis=None)

Rewrite the term in a given basis.

INPUT:

- basis (optional) - either None or a function such that basis(d) is a basis of the weight d multiple zeta values. If None, the Hoffman basis is used.

EXAMPLES:

```sage
z = Multizeta(5) + Multizeta(1,4) + Multizeta(3,2) - 5 * Multizeta(2, 3)
(continues on next page)```
sage: z.simplify_full()
\[-\frac{22}{5}\zeta(2,3) + \frac{12}{5}\zeta(3,2)\]
sage: z.simplify_full(basis=z.parent().basis_filtration)
18\zeta(1,4) - \zeta(5)

sage: z == z.simplify_full() == z.simplify_full(basis=z.parent().basis_filtration)
True

Be careful, that this does not optimize the number of terms:

sage: Multizeta(7).simplify_full()
\frac{352}{151}\zeta(2,2,3) + \frac{672}{151}\zeta(2,3,2) + \frac{528}{151}\zeta(3,2,2)

single_valued()

Return the single-valued version of self.

This is the projection map onto the sub-algebra of single-valued motivic multiple zeta values, as defined by F. Brown in [Bro2013].

This morphism of algebras sends in particular $\zeta(2)$ to 0.

EXAMPLES:

sage: M = Multizetas(QQ)
sage: x = M((2,))
sage: x.single_valued()
0
sage: x = M((3,))
sage: x.single_valued()
2\zeta(3)

algebra_generators(n)

Return a set of multiplicative generators in weight $n$.

This is obtained from hardcoded data, available only up to weight 17.

INPUT:

• $n$ – an integer

EXAMPLES:
sage: M = Multizetas(QQ)
sage: M.algebra_generators(5)
[ζ(5)]
sage: M.algebra_generators(8)
[ζ(3,5)]

an_element()
Return an element of the algebra.

EXAMPLES:

sage: M = Multizetas(QQ)
sage: M.an_element()
ζ() + ζ(1,2) + 1/2*ζ(5)

basis_brown(n)
Return a basis of the algebra of multiple zeta values in weight n.

It was proved by Francis Brown that this is a basis of motivic multiple zeta values.
This is made of all ζ(n₁,...,nᵣ) with parts in {2,3}.

INPUT:
• n – an integer

EXAMPLES:

sage: M = Multizetas(QQ)
sage: M.basis_brown(3)
[ζ(3)]
sage: M.basis_brown(4)
[ζ(2,2)]
sage: M.basis_brown(5)
[ζ(3,2), ζ(2,3)]
sage: M.basis_brown(6)
[ζ(3,3), ζ(2,2,2)]

basis_data(basering, n)
Return an iterator for a basis in weight n.

This is obtained from hardcoded data, available only up to weight 17.

INPUT:
• n – an integer

EXAMPLES:

sage: M = Multizetas(QQ)
sage: list(M.basis_data(QQ, 4))
[4*ζ(1,3) + 2*ζ(2,2)]

basis_filtration(d, reverse=False)
Return a module basis of the homogeneous components of weight d compatible with the length filtration.

INPUT:
• d – (non-negative integer) the weight
• `reverse` – (boolean, default `False`) change the ordering of compositions

EXAMPLES:

```python
sage: M = Multizetas(QQ)
sage: M.basis_filtration(5)
[ζ(5), ζ(1,4)]
sage: M.basis_filtration(6)
[ζ(6), ζ(1,5)]
sage: M.basis_filtration(8)
[ζ(8), ζ(1,7), ζ(2,6), ζ(1,1,6)]
sage: M.basis_filtration(8, reverse=True)
[ζ(8), ζ(6,2), ζ(5,3), ζ(5,1,2)]
sage: M.basis_filtration(0)
[ζ(0)]
sage: M.basis_filtration(1)
[]
```

`degree_on_basis(w)`

Return the degree of the monomial `w`.

This is the sum of terms in `w`.

INPUT:

• `w` – a composition

EXAMPLES:

```python
sage: M = Multizetas(QQ)
sage: x = (2,3)
sage: M.degree_on_basis(x) # indirect doctest
5
```

`half_product(w1, w2)`

Compute half of the product of two elements.

This comes from half of the shuffle product.

**Warning:** This is not a motivic operation.

INPUT:

• `w1, w2` – elements

EXAMPLES:

```python
sage: M = Multizetas(QQ)
sage: M.half_product(M([2]), M([2]))
2*ζ(1,3) + ζ(2,2)
```

`iterated()`

Convert to the algebra of iterated integrals.

This is also available as a method of elements.
EXAMPLES:

```
sage: M = Multizetas(QQ)
sage: x = M((3,2))
sage: M.iterated(3*x)
3*I(10010)
sage: x = M((2,3,2))
sage: M.iterated(4*x)
-4*I(1010010)
```

**iterated_on_basis(w)**

Convert to the algebra of iterated integrals.

Beware that this conversion involves signs in our chosen convention.

**INPUT:**

- `w` – a word

**EXAMPLES:**

```
sage: M = Multizetas(QQ)
sage: x = M.basis().keys()((3,2))
sage: M.iterated_on_basis(x)
I(10010)
sage: x = M.basis().keys()((2,3,2))
sage: M.iterated_on_basis(x)
-I(1010010)
```

**one_basis()**

Return the index of the unit for the algebra.

This is the empty word.

**EXAMPLES:**

```
sage: M = Multizetas(QQ)
sage: M.one_basis()
word:
```

**phi()**

Return the morphism `phi`.

This sends multiple zeta values to the auxiliary F-algebra, which is a shuffle algebra in odd generators $f_3, f_5, f_7, \ldots$ over the polynomial ring in one variable $f_2$.

This is a ring isomorphism, that depends on the choice of a multiplicative basis for the ring of motivic multiple zeta values. Here we use one specific hardcoded basis.

For the precise definition of `phi` by induction, see [Brown2012].

**EXAMPLES:**

```
sage: M = Multizetas(QQ)
sage: m = Multizeta(2,2) + 2*Multizeta(1,3); m
2*ζ(1,3) + ζ(2,2)
sage: M.phi(m)
1/2*f2^2
```

(continues on next page)
sage: Z = Multizeta

sage: B5 = [3*Z(1,4) + 2*Z(2,3) + Z(3,2), 3*Z(1,4) + Z(2,3)]

sage: [M.phi(b) for b in B5]
[-1/2*f5 + f2*f3, 1/2*f5]

example

product_on_basis(w1, w2)

Compute the product of two monomials.

This is done by converting to iterated integrals and using the shuffle product.

INPUT:

• w1, w2 – compositions

EXAMPLES:

sage: M = Multizetas(QQ)

sage: M product_on_basis([2],[2])
4*ζ(1,3) + 2*ζ(2,2)

sage: x = M((2,))

sage: x*x
4*ζ(1,3) + 2*ζ(2,2)

some_elements()

Return some elements of the algebra.

EXAMPLES:

sage: M = Multizetas(QQ)

sage: M some_elements()
(ζ(), ζ(2), ζ(3), ζ(4), ζ(1,2))

class sage.modular.multiple_zeta.Multizetas_iterated(R)

Bases: CombinatorialFreeModule

Secondary class for the algebra of multiple zeta values.

This is used to represent multiple zeta values as iterated integrals of the differential forms \( \omega_0 = dt/t \) and \( \omega_1 = dt/(t - 1) \).

EXAMPLES:

sage: from sage.modular.multiple_zeta import Multizetas_iterated

sage: M = Multizetas_iterated(QQ); M
Algebra of motivic multiple zeta values as convergent iterated integrals over Rational Field

sage: M((1,0))
I(10)

sage: M((1,0))**2
4*I(1100) + 2*I(1010)

sage: M((1,0))*M((1,0,0))
6*I(11000) + 3*I(10100) + I(10010)

D(k)

Return the operator \( D_k \).
INPUT:

• \( k \) – an odd integer, at least 3

EXAMPLES:

```
sage: from sage.modular.multiple_zeta import Multizetas_iterated
sage: M = Multizetas_iterated(QQ)
sage: D3 = M.D(3)
sage: elt = M((1,0,1,0,0)) + 2 * M((1,1,0,0,1,0))
sage: D3(elt)
-6*I(100) # I(110) + 3*I(100) # I(10)
```

\( \text{D}_{on\ basis}(k, w) \)

Return the action of the operator \( D_k \) on the monomial \( w \).

This is one main tool in the procedure that allows to map the algebra of multiple zeta values to the F Ring.

INPUT:

• \( k \) – an odd integer, at least 3

• \( w \) – a word in 0 and 1

EXAMPLES:

```
sage: from sage.modular.multiple_zeta import Multizetas_iterated
sage: M = Multizetas_iterated(QQ)
sage: M.D_on_basis(3,(1,1,1,0,0))
I(110) # I(10) + 2*I(100) # I(10)
sage: M.D_on_basis(3,(1,0,1,0,0))
3*I(100) # I(10)
sage: M.D_on_basis(5,(1,0,0,0,1,0,0,1,0,0))
10*I(10000) # I(10100)
```

**class Element**

**Bases:** IndexedFreeModuleElement

**composition()**

Convert to the algebra of multiple zeta values of composition style.

This means the algebra \( \text{Multizetas} \).

EXAMPLES:

```
sage: from sage.modular.multiple_zeta import Multizetas_iterated
sage: M = Multizetas_iterated(QQ)
sage: x = M((1,0,1,0))
sage: x.composition()
\zeta(2,2)
sage: x = M((1,0,1,0,0))
sage: x.composition()
\zeta(2,3)
sage: x = M((1,0,1,0,0,1,0))
sage: x.composition()
-\zeta(2,3,2)
```
coproduct()

Return the coproduct of self.

EXAMPLES:

```python
sage: from sage.modular.multiple_zeta import Multizetas_iterated
sage: M = Multizetas_iterated(QQ)
sage: a = 3*Multizeta(1,3) + Multizeta(2,3)
sage: a.iterated().coproduct()
3*I() # I(1100) + I() # I(10100) + I(10100) # I() + 3*I(100) # I(10)
```

is_zero()

Return whether this element is zero.

EXAMPLES:

```python
sage: from sage.modular.multiple_zeta import Multizetas_iterated
sage: M = Multizetas_iterated(QQ)
sage: M(0).is_zero()
True
sage: M(1).is_zero()
False
sage: (M((1,1,0)) - M((1,0,0))).is_zero()
True
```

umerical_approx(prec=None, digits=None, algorithm=None)

Return a numerical approximation as a sage real.

EXAMPLES:

```python
sage: from sage.modular.multiple_zeta import Multizetas_iterated
sage: M = Multizetas_iterated(QQ)
sage: x = M((1,0,1,0))
sage: y = M((1, 0, 0))

sage: (3*x+y).n()  # indirect doctest
1.23317037269047
```

phi()

Return the image of self by the morphism phi.

This sends multiple zeta values to the auxiliary F-algebra.

EXAMPLES:

```python
sage: from sage.modular.multiple_zeta import Multizetas_iterated
sage: M = Multizetas_iterated(QQ)
sage: M((1,1,0)).phi()
f3
```

simplify()

Gather terms using the duality relations.

This can help to lower the number of monomials.

EXAMPLES:
sage: from sage.modular.multiple_zeta import Multizetas_iterated
sage: M = Multizetas_iterated(QQ)
sage: z = 4*M((1,0,0)) + 3*M((1,1,0))
sage: z.simplify()
I(100)

composition()
Convert to the algebra of multiple zeta values of composition style.
This means the algebra \textit{Multizetas}.
This is also available as a method of elements.
EXAMPLES:

sage: from sage.modular.multiple_zeta import Multizetas_iterated
sage: M = Multizetas_iterated(QQ)
sage: x = M((1,0))
sage: M.composition(2*x)
-2*\zeta(2)
sage: x = M((1,0,1,0,0))
sage: M.composition(x)
\zeta(2,3)

composition_on_basis(w, basering=None)
Convert to the algebra of multiple zeta values of composition style.
INPUT:
• basering – optional choice of the coefficient ring
EXAMPLES:

sage: from sage.modular.multiple_zeta import Multizetas_iterated
sage: M = Multizetas_iterated(QQ)
sage: x = Word((1,0,1,0,0))
sage: M.composition_on_basis(x)
\zeta(2,3)
sage: x = Word((1,0,1,0,0,1,0))
sage: M.composition_on_basis(x)
-\zeta(2,3,2)

coproduct()
Return the motivic coproduct of an element.
EXAMPLES:

sage: from sage.modular.multiple_zeta import Multizetas_iterated
sage: M = Multizetas_iterated(QQ)
sage: a = 3*Multizeta(1,4) + Multizeta(2,3)
sage: M.coproduct(a.iterated())
3*I() # I(11000) + I() # I(10100) + 3*I(11000) # I() + I(10100) # I()

coproduct_on_basis(w)
Return the motivic coproduct of a monomial.
EXAMPLES:

```python
sage: from sage.modular.multiple_zeta import Multizetas_iterated
sage: M = Multizetas_iterated(QQ)
sage: M.coproduct_on_basis([1,0])
I() # I(10)
sage: M.coproduct_on_basis((1,0,1,0))
I() # I(1010)
```

**degree_on_basis**($w$)

Return the degree of the monomial $w$.

This is the length of the word.

**INPUT:**

- $w$ – a word in 0 and 1

**EXAMPLES:**

```python
sage: from sage.modular.multiple_zeta import Multizetas_iterated
sage: M = Multizetas_iterated(QQ)
sage: x = Word((1,0,1,0,0))
sage: M.degree_on_basis(x)
5
```

**dual_on_basis**($w$)

Return the order of the word and exchange letters 0 and 1.

This is an involution.

**INPUT:**

- $w$ – a word in 0 and 1

**EXAMPLES:**

```python
sage: from sage.modular.multiple_zeta import Multizetas_iterated
sage: M = Multizetas_iterated(QQ)
sage: x = Word((1,0,1,0,0))
sage: M.dual_on_basis(x)
-I(11010)
```

**half_product**()

Compute half of the product of two elements.

This is half of the shuffle product.

**Warning:** This is not a motivic operation.

**INPUT:**

- $w1, w2$ – elements

**EXAMPLES:**
```python
sage: from sage.modular.multiple_zeta import Multizetas_iterated
sage: M = Multizetas_iterated(QQ)
sage: x = M(Word([1,0]))
sage: M.half_product(x,x)
2*I(1100) + I(1010)
```

**half_product_on_basis** *(w1, w2)*

Compute half of the product of two monomials.

This is half of the shuffle product.

**Warning:** This is not a motivic operation.

**INPUT:**

• *w1, w2* – monomials

**EXAMPLES:**

```python
sage: from sage.modular.multiple_zeta import Multizetas_iterated
sage: M = Multizetas_iterated(QQ)
sage: x = Word([1,0])
sage: M.half_product_on_basis(x,x)
2*I(1100) + I(1010)
```

**one_basis()**

Return the index of the unit for the algebra.

This is the empty word.

**EXAMPLES:**

```python
sage: from sage.modular.multiple_zeta import Multizetas_iterated
sage: M = Multizetas_iterated(QQ)
sage: M.one_basis()
word: phi()

Return the morphism phi.

This sends multiple zeta values to the auxiliary F-algebra.

**EXAMPLES:**

```python
sage: from sage.modular.multiple_zeta import Multizetas_iterated
sage: M = Multizetas_iterated(QQ)
sage: m = Multizeta(1,0,1,0) + 2*Multizeta(1,1,0,0); m
2*I(1100) + I(1010)
sage: M.phi(m)
1/2*f2^2
sage: Z = Multizeta
sage: B5 = [3*Z(1,4) + 2*Z(2,3) + Z(3,2), 3*Z(1,4) + Z(2,3)]
sage: [M.phi(b.iterated()) for b in B5]
[-1/2*f5 + f2*f3, 1/2*f5]
```

(continues on next page)
\begin{verbatim}
  sage: B6 = [6*Z(1,5) + 3*Z(2,4) + Z(3,3),
          6*Z(1,1,4) + 4*Z(1,2,3) + 2*Z(1,3,2) + 2*Z(2,1,3) + Z(2,2,2)]
  sage: [M.phi(b.iterated()) for b in B6]
  [f3f3, 1/6*f2^3]
\end{verbatim}

**phi\_extended(\textit{w})**

Return the image of the monomial \textit{w} by the morphism \textit{phi}.

**INPUT:**

\begin{itemize}
  \item \textit{w} – a word in 0 and 1
\end{itemize}

**OUTPUT:**

an element in the auxiliary F-algebra

The coefficients are in the base ring.

**EXAMPLES:**

\begin{verbatim}
  sage: from sage.modular.modform.constructor import Multizetas_iterated
  sage: M = Multizetas_iterated(QQ)
  sage: M.phi_extended((1,0))
  -f2
  sage: M.phi_extended((1,0,0))
  -f3
  sage: M.phi_extended((1,1,0))
  f3
  sage: M.phi_extended((1,0,1,0,0))
  -11/2*f5 + 3*f2*f3
\end{verbatim}

More complicated examples:

\begin{verbatim}
  sage: from sage.modular.modform.constructor import composition_to_iterated
  sage: M.phi_extended(composition_to_iterated((4,3)))
  -18*f7 + 10*f2*f5 + 2/5*f2^2*f3
  sage: M.phi_extended(composition_to_iterated((3,4)))
  17*f7 - 10*f2*f5
\end{verbatim}

**product\_on\_basis(\textit{w1}, \textit{w2})**

Compute the product of two monomials.

This is the shuffle product.

**INPUT:**
• w1, w2 – words in 0 and 1

EXAMPLES:

```python
sage: from sage.modular.multiple_zeta import Multizetas_iterated
sage: M = Multizetas_iterated(QQ)
sage: x = Word([1,0])
sage: M.product_on_basis(x,x)
4*I(1100) + 2*I(1010)
sage: y = Word([1,1,0])
sage: M.product_on_basis(y,x)
I(10110) + 3*I(11010) + 6*I(11100)
```

`sage.modular.multiple_zeta.coeff_phi(w)`
Return the coefficient of $f_k$ in the image by $\phi$.

INPUT:

• w – a word in 0 and 1 with $k$ letters (where $k$ is odd)

OUTPUT:
a rational number

EXAMPLES:

```python
sage: from sage.modular.multiple_zeta import coeff_phi
sage: coeff_phi(Word([1,0,0]))
-1
sage: coeff_phi(Word([1,1,0]))
1
sage: coeff_phi(Word([1,1,0,1,0]))
11/2
sage: coeff_phi(Word([1,1,0,0,0,1,0]))
109/16
```

`sage.modular.multiple_zeta.composition_to_iterated(w, reverse=False)`
Convert a composition to a word in 0 and 1.

By default, the chosen convention maps (2,3) to (1,0,1,0,0), respecting the reading order from left to right.

The inverse map is given by `iterated_to_composition()`.

EXAMPLES:

```python
sage: from sage.modular.multiple_zeta import composition_to_iterated
sage: composition_to_iterated((1,2))
(1, 1, 0)
sage: composition_to_iterated((3,1,2))
(1, 0, 0, 1, 1, 0)
sage: composition_to_iterated((3,1,2,4))
(1, 0, 0, 1, 1, 0, 1, 0, 0, 0)
```

`sage.modular.multiple_zeta.compute_u_on_basis(w)`
Compute the value of $u$ on a multiple zeta value.

INPUT:

• w – a word in 0,1
 OUTPUT:

an element of $F_{\text{ring}}(\mathbb{Q})$

EXAMPLES:

```python
sage: from sage.modular.multiple_zeta import compute_u_on_basis
sage: compute_u_on_basis((1,0,0,0,1,0))
-2*f3f3
sage: compute_u_on_basis((1,1,0,0))
f2*f3
sage: compute_u_on_basis((1,0,1,0,0,1,0))
11/2*f2*f5
sage: compute_u_on_basis((1,0,0,1,0,0,0,0))
-5*f5f3
sage: compute_u_on_basis((1,0,0,1,0,1,0,0,1,0))
75/4*f3f7 + 81/4*f5f5 + 75/8*f7f3 + 11*f2*f3f5 - 9*f2*f5f3
```

```
sage.modular.multiple_zeta.compute_u_on_compo(compo)
Compute the value of the map $u$ on a multiple zeta value.
INPUT:

• compo – a composition

OUTPUT:

an element of $F_{\text{ring}}(\mathbb{Q})$

EXAMPLES:

```python
sage: from sage.modular.multiple_zeta import compute_u_on_compo
sage: compute_u_on_compo((2,4))
2*f3f3
sage: compute_u_on_compo((2,3,2))
-11/2*f2*f5
sage: compute_u_on_compo((3,2,3,2))
-75/4*f3f7 + 81/4*f5f5 + 75/8*f7f3 + 11*f2*f3f5 - 9*f2*f5f3
```

```
sage.modular.multiple_zeta.coproduct_iterator(paire)
Return an iterator for terms in the coproduct.
This is an auxiliary function.

INPUT:

• paire – a pair (list of indices, end of word)

OUTPUT:

iterator for terms in the motivic coproduct

Each term is seen as a list of positions.

EXAMPLES:

```
sage: from sage.modular.multiple_zeta import coproduct_iterator
sage: list(coproduct_iterator(([0],[0,1,0,1])))
[[0, 1, 2, 3]]
sage: list(coproduct_iterator(([0],[0,1,0,1,1,0,1])))
[[0, 1, 2, 3, 4, 5, 6], [0, 1, 2, 6], [0, 1, 5, 6], [0, 4, 5, 6], [0, 6]]

sage.modular.multiple_zeta.dual_composition(c)

Return the dual composition of c.

This is an involution on compositions such that associated multizetas are equal.

INPUT:

- c – a composition

OUTPUT:

a composition

EXAMPLES:

sage: from sage.modular.multiple_zeta import dual_composition
sage: dual_composition([3])
(1, 2)
sage: dual_composition(dual_composition([3,4,5])) == (3,4,5)
True

sage.modular.multiple_zeta.extend_multiplicative_basis(B, n)

Extend a multiplicative basis into a basis.

This is an iterator.

INPUT:

- B – function mapping integer to list of tuples of compositions
- n – an integer

OUTPUT:

Each term is a tuple of tuples of compositions.

EXAMPLES:

sage: from sage.modular.multiple_zeta import extend_multiplicative_basis
sage: from sage.modular.multiple_zeta import B_data
sage: list(extend_multiplicative_basis(B_data,5))
[((5,),), ((3,), (2,))]
sage: list(extend_multiplicative_basis(B_data,6))
[((3,), (3,), (2,), (2,)), (2,)]
sage: list(extend_multiplicative_basis(B_data,7))
[((7,),), ((5,), (2,)), ((3,), (2,), (2,))]

sage.modular.multiple_zeta.iterated_to_composition(w, reverse=False)

Convert a word in 0 and 1 to a composition.

By default, the chosen convention maps (1,0,1,0,0) to (2,3).

The inverse map is given by composition_to_iterated().

EXAMPLES:
```python
sage: from sage.modular.multiple_zeta import iterated_to_composition
sage: iterated_to_composition([1,0,1,0,0])
(2, 3)
sage: iterated_to_composition(Word([1,1,0]))
(1, 2)
sage: iterated_to_composition(Word([1,1,0,1,0,0]))
(1, 2, 1, 3)
```

```python
sage.modular.multiple_zeta.minimize_term(w, cf)
```

Return the largest among \( w \) and the dual word of \( w \).

**INPUT:**

- \( w \) – a word in the letters 0 and 1
- \( cf \) – a coefficient

**OUTPUT:**

(\text{word}, \text{coefficient})

The chosen order is lexicographic with 1 < 0.

If the dual word is chosen, the sign of the coefficient is changed, otherwise the coefficient is returned unchanged.

**EXAMPLES:**

```python
sage: from sage.modular.multiple_zeta import minimize_term, Words10
sage: minimize_term(Words10((1,1,0)), 1)
(word: 100, -1)
sage: minimize_term(Words10((1,0,0)), 1)
(word: 100, 1)
```

```python
sage.modular.multiple_zeta.phi_on_basis(L)
```

Compute the value of \( \phi \) on the hardcoded basis.

**INPUT:**

a list of compositions, each composition in the hardcoded basis

This encodes a product of multiple zeta values.

**OUTPUT:**

an element in \( F\text{-}_\text{ring}() \)

**EXAMPLES:**

```python
sage: from sage.modular.multiple_zeta import phi_on_basis
sage: phi_on_basis([(3,), (3,)])
2*f3f3
sage: phi_on_basis([(2,), (2,)])
f2^2
sage: phi_on_basis([(2,), (3,), (3,)])
2*f2*f3f3
```

```python
sage.modular.multiple_zeta.phi_on_multiplicative_basis(compo)
```

Compute \( \phi \) on one single multiple zeta value.

**INPUT:**

4.17. Algebra of motivic multiple zeta values 479
• compo – a composition (in the hardcoded multiplicative base)

OUTPUT:

an element in F_ring() with rational coefficients

EXAMPLES:

```sage
from sage.modular.multiple_zeta import phi_on_multiplicative_basis
sage: phi_on_multiplicative_basis((2,))
f2
sage: phi_on_multiplicative_basis((3,))
f3
```

`sage.modular.multiple_zeta.rho_inverse(elt)`

Return the image by the inverse of rho.

INPUT:

• elt – an homogeneous element of the F ring

OUTPUT:

a linear combination of multiple zeta values

EXAMPLES:

```sage
from sage.modular.multiple_zeta import rho_inverse
from sage.modular.multiple_zeta_F_algebra import F_algebra
sage: A = F_algebra(QQ)
sage: f = A.gen
sage: rho_inverse(f(3))
ζ(3)
sage: rho_inverse(f(9))
ζ(9)
sage: rho_inverse(A("53"))
-1/5*ζ(3, 5)
```

`sage.modular.multiple_zeta.rho_matrix_inverse()`

Return the matrix of the inverse of rho.

This is the matrix in the chosen bases, namely the hardcoded basis of multiple zeta values and the natural basis of the F ring.

INPUT:

• n – an integer

EXAMPLES:

```sage
from sage.modular.multiple_zeta import rho_matrix_inverse
sage: rho_matrix_inverse(3)
[1]
sage: rho_matrix_inverse(8)
[-1/5 0 0 0]
[ 1/5 1 0 0]
[ 0 0 1/2 0]
[ 0 0 0 1]
```
CHAPTER
FIVE

INDICES AND TABLES

• Index
• Module Index
• Search Page


<table>
<thead>
<tr>
<th>Module</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>sage.modular.buzzard</td>
<td>335</td>
</tr>
<tr>
<td>sage.modular.cusps</td>
<td>323</td>
</tr>
<tr>
<td>sage.modular.cusps_nf</td>
<td>428</td>
</tr>
<tr>
<td>sage.modular.dims</td>
<td>329</td>
</tr>
<tr>
<td>sage.modular.dirichlet</td>
<td>295</td>
</tr>
<tr>
<td>sage.modular.etaproducts</td>
<td>368</td>
</tr>
<tr>
<td>sage.modular.hypergeometric_motive</td>
<td>438</td>
</tr>
<tr>
<td>sage.modular.local_comp.liftings</td>
<td>365</td>
</tr>
<tr>
<td>sage.modular.local_comp.local_comp</td>
<td>336</td>
</tr>
<tr>
<td>sage.modular.local_comp.smoothchar</td>
<td>346</td>
</tr>
<tr>
<td>sage.modular.local_comp.type_space</td>
<td>361</td>
</tr>
<tr>
<td>sage.modular.modform.ambient</td>
<td>22</td>
</tr>
<tr>
<td>sage.modular.modform.ambient_eps</td>
<td>28</td>
</tr>
<tr>
<td>sage.modular.modform.ambient_g0</td>
<td>31</td>
</tr>
<tr>
<td>sage.modular.modform.ambient_gl</td>
<td>32</td>
</tr>
<tr>
<td>sage.modular.modform.ambient_R</td>
<td>33</td>
</tr>
<tr>
<td>sage.modular.modform.constructor</td>
<td>1</td>
</tr>
<tr>
<td>sage.modular.modform.cuspidal_submodule</td>
<td>35</td>
</tr>
<tr>
<td>sage.modular.modform.eis_series</td>
<td>45</td>
</tr>
<tr>
<td>sage.modular.modform.eis_series_cython</td>
<td>47</td>
</tr>
<tr>
<td>sage.modular.modform.eisenstein_submodule</td>
<td>38</td>
</tr>
<tr>
<td>sage.modular.modform.element</td>
<td>48</td>
</tr>
<tr>
<td>sage.modular.modform.half_integral</td>
<td>85</td>
</tr>
<tr>
<td>sage.modular.modform.hecke_operator_on_qexp</td>
<td>78</td>
</tr>
<tr>
<td>sage.modular.modform.j_invariant</td>
<td>96</td>
</tr>
<tr>
<td>sage.modular.modform.notes</td>
<td>98</td>
</tr>
<tr>
<td>sage.modular.modform.numerical</td>
<td>79</td>
</tr>
<tr>
<td>sage.modular.modform.ring</td>
<td>87</td>
</tr>
<tr>
<td>sage.modular.modform.space</td>
<td>6</td>
</tr>
<tr>
<td>sage.modular.modform.submodule</td>
<td>34</td>
</tr>
<tr>
<td>sage.modular.modform.theta</td>
<td>97</td>
</tr>
<tr>
<td>sage.modular.modform.vm_basis</td>
<td>83</td>
</tr>
<tr>
<td>sage.modular.modform.hecketriangle.abstract_ring</td>
<td>120</td>
</tr>
<tr>
<td>sage.modular.modform.hecketriangle.abstract_space</td>
<td>146</td>
</tr>
<tr>
<td>sage.modular.modform.hecketriangle.analytic_type</td>
<td>253</td>
</tr>
<tr>
<td>sage.modular.modform.hecketriangle.constructor</td>
<td>197</td>
</tr>
<tr>
<td>sage.modular.modform.hecketriangle.element</td>
<td>171</td>
</tr>
<tr>
<td>sage.modular.modform.hecketriangle.functors</td>
<td>201</td>
</tr>
<tr>
<td>sage.modular.modform.hecketriangle.graded_ring</td>
<td>257</td>
</tr>
<tr>
<td>sage.modular.modform.hecketriangle.graded_ring_element</td>
<td>174</td>
</tr>
<tr>
<td>sage.modular.modform.hecketrianglehecke_triangle_group_element</td>
<td>218</td>
</tr>
<tr>
<td>sage.modular.modform.hecketriangle.hecke_triangle_groups</td>
<td>204</td>
</tr>
<tr>
<td>sage.modular.modform.hecketriangle.readme</td>
<td>99</td>
</tr>
<tr>
<td>sage.modular.modform.hecketriangle.series_constructor</td>
<td>270</td>
</tr>
<tr>
<td>sage.modular.modform.hecketriangle.space</td>
<td>258</td>
</tr>
<tr>
<td>sage.modular.modform.hecketriangle.subspace</td>
<td>266</td>
</tr>
<tr>
<td>sage.modular.modform_quasimodform.element</td>
<td>286</td>
</tr>
<tr>
<td>sage.modular.modform_quasimodform.ring</td>
<td>277</td>
</tr>
<tr>
<td>sage.modular.quatalg.brandt</td>
<td>417</td>
</tr>
<tr>
<td>sage.modular.ssmod.ssmod</td>
<td>407</td>
</tr>
<tr>
<td>sage.modular.multiple_zeta</td>
<td>455</td>
</tr>
<tr>
<td>sage.modular.overconvergent.genus0</td>
<td>382</td>
</tr>
<tr>
<td>sage.modular.overconvergent.hecke_series</td>
<td>395</td>
</tr>
<tr>
<td>sage.modular.overconvergent.weightspace</td>
<td>376</td>
</tr>
<tr>
<td>sage.modular.quasimodform.element</td>
<td>286</td>
</tr>
<tr>
<td>sage.modular.quasimodform.ring</td>
<td>277</td>
</tr>
<tr>
<td>sage.modular.quatalg.brandt</td>
<td>417</td>
</tr>
<tr>
<td>sage.modular.ssmod.ssmod</td>
<td>407</td>
</tr>
<tr>
<td>A</td>
<td>(sage.modular.modform_hecketriangle.hecke_triangle_group_element.HeckeTriangleGroupElement method), 218</td>
</tr>
<tr>
<td>----------</td>
<td>---------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>abelian_variance()</td>
<td>(sage.modular.modform.element.Newform method), 69</td>
</tr>
<tr>
<td>ABmatrix()</td>
<td>(sage.modular.cusps_nf.NFCusp method), 432</td>
</tr>
<tr>
<td>acton()</td>
<td>(sage.modular.modform_hecketriangle.hecke_triangle_group_element.HeckeTriangleGroupElement method), 218</td>
</tr>
<tr>
<td>additive_order()</td>
<td>(sage.modular.overconvergent.genus0.OverconvergentModularFormElement method), 384</td>
</tr>
<tr>
<td>algebra_generators()</td>
<td>(sage.modular.multiple_zeta.Multizetas method), 466</td>
</tr>
<tr>
<td>AlgebraicWeight</td>
<td>(class in sage.modular.overconvergent.weightspace, 378</td>
</tr>
<tr>
<td>All_iterated</td>
<td>(class in sage.modular.multiple_zeta, 458</td>
</tr>
<tr>
<td>All_iterated.Element</td>
<td>(class in sage.modular.multiple_zeta, 458</td>
</tr>
<tr>
<td>AllCusps()</td>
<td>(in module sage.modular.etaproducts), 368</td>
</tr>
<tr>
<td>alpha()</td>
<td>(sage.modular.hypergeometric_motive.HypergeometricData method), 441</td>
</tr>
<tr>
<td>alpha()</td>
<td>(sage.modular.modform_hecketriangle.hecke_triangle_groups.HeckeTriangleGroup method), 207</td>
</tr>
<tr>
<td>alpha_beta()</td>
<td>(sage.modular.hypergeometric_motive.HypergeometricData method), 441</td>
</tr>
<tr>
<td>alpha_to_cyclotomic()</td>
<td>(in module sage.modular.hypergeometric_motive), 452</td>
</tr>
<tr>
<td>ambient_coordinate_vector()</td>
<td>(sage.modular.modform_hecketriangle.abstract_space.FormsSpace_abstract method), 151</td>
</tr>
<tr>
<td>ambient_coordinate_vector()</td>
<td>(sage.modular.modform_hecketriangle.element.FormsElement method), 171</td>
</tr>
<tr>
<td>ambient_module()</td>
<td>(sage.modular.modform_hecketriangle.abstract_space.FormsSpace_abstract method), 152</td>
</tr>
<tr>
<td>ambient_space()</td>
<td>(sage.modular.modform.ambient.ModularFormsAmbient method), 22</td>
</tr>
<tr>
<td>ambient_space()</td>
<td>(sage.modular.modform_hecketriangle.abstract_space.FormsSpace_abstract method), 152</td>
</tr>
<tr>
<td>an_element()</td>
<td>(sage.modular.multiple_zeta.Multizetas method), 256</td>
</tr>
<tr>
<td>analytic_name()</td>
<td>(sage.modular.modform_hecketriangle.analytic_type.AnalyticTypeElement method), 255</td>
</tr>
<tr>
<td>analytic_space_name()</td>
<td>(sage.modular.modform_hecketriangle.analytic_type.AnalyticTypeElement method), 256</td>
</tr>
<tr>
<td>AnalyticType</td>
<td>(class in sage.modular.modform_hecketriangle.analytic_type, 253</td>
</tr>
<tr>
<td>AnalyticType</td>
<td>(class in sage.modular.modform_hecketriangle.abstract_ring.FormsRing_abstract, 120</td>
</tr>
<tr>
<td>AnalyticType</td>
<td>(class in sage.modular.modform_hecketriangle.functors.FormsRingFunctor, 201</td>
</tr>
<tr>
<td>AnalyticType</td>
<td>(class in sage.modular.modform_hecketriangle.functors.FormsSpaceFunctor, 202</td>
</tr>
<tr>
<td>AnalyticType</td>
<td>(class in sage.modular.modform_hecketriangle.graded_ring_element.FormsRingElement, 174</td>
</tr>
<tr>
<td>ArbitraryWeight</td>
<td>(class in sage.modular.modform_hecketriangle.analytic_type, 254</td>
</tr>
<tr>
<td>apply()</td>
<td>(sage.modular.cusps.Cusp method), 323</td>
</tr>
<tr>
<td>apply()</td>
<td>(sage.modular.cusps_nf.NFCusp method), 432</td>
</tr>
<tr>
<td>as_hyperbolic_plane_isometry()</td>
<td>(sage.modular.modform_hecketriangle.hecke_triangle_group_element.HeckeTriangleGroupElement method), 220</td>
</tr>
<tr>
<td>as_ring_element()</td>
<td>(sage.modular.modform_hecketriangle.graded_ring_element.FormsRingElement method), 475</td>
</tr>
<tr>
<td>AT</td>
<td>(sage.modular.modform_hecketriangle.abstract_ring.FormsRing_abstract, 120</td>
</tr>
<tr>
<td>AT</td>
<td>(sage.modular.modform_hecketriangle.functors.FormsRingFunctor, 201</td>
</tr>
</tbody>
</table>

487
Index 489
coordinates() (sage.modular.overconvergent.genus0.OverconvergentModularFormsSpace method), 384
coproduct() (sage.modular.multiple_zeta.Multizetas_iterated.CuspidalSubmodule (class in sage.modular.modform.cuspidal_submodule), 37
coproduct() (sage.modular.multiple_zeta.Multizetas_iterated.Element method), 472
coproduct() (sage.modular.multiple_zeta.Multizetas_iterated.Element method), 470
coproduct_iterator() (in module sage.modular.multiple_zeta), 477
coproduct_on_basis() (sage.modular.multiple_zeta.Multizetas_iterated method), 472
cps_u() (sage.modular.overconvergent.genus0.OverconvergentModularFormsSpace method), 389
create_key() (sage.modular.dirichlet.DirichletGroupFactory method), 316
create_object() (sage.modular.dirichlet.DirichletGroupFactory method), 316
Cusp (class in sage.modular.cusps), 323
CuspFamily (class in sage.modular.etaproducts), 369
CuspForms (class in sage.modular.modform_hecketriangle.space), 258
CuspForms() (in module sage.modular.modform.constructor), 1
CuspFormsRing (class in sage.modular.modform_hecketriangle.graded_ring), 257
cuspidal_ideal_generators() (sage.modular.modform_hecketriangle.hecke_triangle_group_element method), 252
cuspidal_submodule() (class in sage.modular.modform.cuspidal_submodule), 35
CuspidalSubmodule (class in sage.modular.modform.cuspidal_submodule), 37
CuspidalSubmodule_eps (class in sage.modular.modform.cuspidal_submodule), 37
CuspidalSubmodule_g0_Q (class in sage.modular.modform.cuspidal_submodule), 37
CuspidalSubmodule_g1_Q (class in sage.modular.modform.cuspidal_submodule), 37
CuspidalSubmodule_level1_Q (class in sage.modular.modform.cuspidal_submodule), 37
CuspidalSubmodule_modsym_qexp (class in sage.modular.modform.cuspidal_submodule), 37
CuspidalSubmodule_R (class in sage.modular.modform.cuspidal_submodule), 37
CuspidalSubmodule_wt1_eps (class in sage.modular.modform.cuspidal_submodule), 37
CuspidalSubmodule_wt1_gH (class in sage.modular.modform.cuspidal_submodule), 37
CuspidalSubmodule_gH_Q (class in sage.modular.modform.cuspidal_submodule), 37
CuspidalSubmodule_level1_gH (class in sage.modular.modform.cuspidal_submodule), 37
CuspidalSubmodule_wt1_gH (class in sage.modular.modform.cuspidal_submodule), 37
cyclic_representative() (in module sage.modular.modform_hecketriangle.hecke_triangle_group_element method), 252
cyclic_submodules() (sage.modular.quatalg.brandt.BrandtModule_class method), 422
cyclotomic_data() (sage.modular.hypergeometric_motive.HypergeometricData method), 442
cyclotomic_restriction() (in module sage.modular.modform_hecketriangle.hecke_triangle_group_element method), 43
cyclotomic_restriction_tower() (in module sage.modular.modform_hecketriangle.hecke_triangle_group_element method), 44
cyclotomic_submodule() (sage.modular.modform.ambient_1.ModularFormsAmbient method), 32
cyclotomic_submodule() (sage.modular.modform.ambient_eps.ModularFormsAmbient method), 39
cyclotomic_submodule() (sage.modular.modform.ambient_g0.ModularFormsAmbient method), 31
cyclotomic_submodule() (sage.modular.modform.ambient_g1.ModularFormsAmbient method), 32
cyclotomic_submodule() (sage.modular.modform.ambient_R.ModularFormsAmbient method), 33
cyclotomic_submodule() (sage.modular.modform.ambient_gH.Q method), 34

cyclotomic_alpha() (in module sage.modular.hypergeometric_motive), 453
cyclotomic_to_alpha() (in module sage.modular.hypergeometric_motive), 454

d() (sage.modular.modform_hecketriangle.hecke_triangle_group_element method), 8

d() (sage.modular.modform_hecketriangle.hecke_triangle_group_element method), 88

d() (sage.modular.modform_space.ModularFormsSpace method), 8

Index 491
sage.modular.modform.eisenstein_submodule), 40
E_k_ZZ() (in module sage.modular.modform.eis_series_cython), 47
Element (sage.modular.cusps.Cusps_class attribute), 328
Element (sage.modular.dirichlet.DirichletGroup_class attribute), 316
Element (sage.modular.etaproducts.EtaGroup_class attribute), 372
Element (sage.modular.local_comp.smoothchar.SmoothCharacterGroupQuadratic attribute), 348
Element (sage.modular.modform_ring.ModularFormsRing attribute), 88
Element (sage.modular.modform.space.ModularFormsSpace attribute), 7
Element (sage.modular.modform_hecketriangle.abstract_ring.FormsRing_abstract attribute), 127
Element (sage.modular.modform_hecketriangle.abstract_space.FormsSpace_abstract attribute), 146
Element (sage.modular.modform_hecketriangle.analytic_type.AnalyticType attribute), 254
Element (sage.modular.modform_hecketriangle.hecke_triangle_groups.HeckeTriangleGroup attribute), 204
Element (sage.modular.quasimodform.ring.QuasiModularForms attribute), 280
Element (sage.modular.quatalg.brandt.BrandtModule_class attribute), 420
element() (sage.modular.dirichlet.DirichletCharacter method), 299
element() (sage.modular.modform.element.Newform method), 73
element_from_ambient_coordinates() (sage.modular.modform_hecketriangle.abstract_space.FormsSpace abstract method), 158
element_from_coordinates() (sage.modular.modform_hecketriangle.abstract_space.FormsSpace abstract method), 159
element_repr_method() (sage.modular.modform_hecketriangle.hecke_triangle_groups.HeckeTriangleGroup method), 210
elliptic_curve() (sage.modular.modform.element.ModularFormElement method), 58
embedded_submodule() (sage.modular.modform.space.ModularFormsSpace method), 12
enumerate_hypergeometric_data() (in module sage.modular.hypergeometric_motive), 454
ep() (sage.modular.modform_hecketriangle.abstract_space.FormsSpace abstract method), 159
ep() (sage.modular.modform_hecketriangle.graded_ring_element.FormsRingElement method), 179
eta_poly_relations() (in module sage.modular.etaproducts), 374
EtaGroup() (in module sage.modular.etaproducts), 369
EtaGroup_class (class in sage.modular.etaproducts), 372
EtaGroupElement (class in sage.modular.etaproducts), 370
EtaProduct() (in module sage.modular.etaproducts), 374
euler_factor() (sage.modular.hypergeometric_motive.HypergeometricData method), 443
evaluate() (sage.modular.modform_hecketriangle.graded_ring_element.FormsRingElement method), 79
example_type_space() (in module sage.modular.local_comp.type_space), 364
expand() (sage.modular.linalg.liquidation.HalfIntegralLiquidation method), 459
expand_on_basis() (sage.modular.multiple_zeta.All_iterated method), 459
exponents() (sage.modular.dirichlet.DirichletGroup_class method), 136
F
F_basis() (sage.modular.modform_hecketriangle.abstract_space.FormsSpace abstract method), 146
F_basis_Ek() (sage.modular.modform_hecketriangle.abstract_space.FormsSpace abstract method), 147
f_i() (sage.modular.modform_hecketriangle.abstract_ring.FormsRingElement method), 133
f_i_ZZ() (sage.modular.modform_hecketriangle.series_constructor.MFSeriesConstructor method), 273
f_inf() (sage.modular.modform_hecketriangle.abstract_ring.FormsRingElement method), 148
f_inf() (sage.modular.modform_hecketriangle.abstract_ring.FormsRingElement method), 149
f_i() (sage.modular.modform_hecketriangle.abstract_ring.FormsRingElement method), 133
f_i_ZZ() (sage.modular.modform_hecketriangle.series_constructor.MFSeriesConstructor method), 273
f_rho() (sage.modular.modform_hecketriangle.abstract_ring.FormsRingElement method), 136
Index 495

f_rho_ZZ() (sage.modular.modform_hecketriangle.series_constructor.MFSeriesConstructor method), 274
F_simple() (sage.modular.modform_hecketriangle.abstract_space.FormsSpace_abstract method), 149
Faber_pol() (sage.modular.modform_hecketriangle.abstract_space.FormsSpace_abstract method), 149
free_module() (sage.modular.local_comp.type_space.TypeSpace method), 361
find_in_space() (sage.modular.local_comp.type_space, 364
fixed_field() (sage.modular.dirichlet.DirichletCharacter, 299
fixed_field_polynomial() (sage.modular.dirichlet.DirichletCharacter method), 300
fixed_points() (sage.modular.modform_hecketriangle.hecke_triangle_group.element.HeckeTriangleGroupElement method), 226
form() (sage.modular.local_comp.type_space.TypeSpace method), 361
FormsElement (class in sage.modular.modform_hecketriangle.element), 171
FormsElement (sage.modular.modform_hecketriangle.abstract_space.FormsSpace_abstract attribute), 151
FormsRing (class in module sage.modular.modform_hecketriangle.constructor), 197
FormsRing_abstract (class in sage.modular.modform_hecketriangle.abstract_ring), 120
FormsRingElement (class in sage.modular.modform_hecketriangle.graded_ring_element.attribute), 127
FormsRingFunctor (class in sage.modular.modform_hecketriangle.functors), 201
FormsSpace (class in module sage.modular.modform_hecketriangle.constructor), 198
FormsSpace_abstract (class in sage.modular.modform_hecketriangle.abstract_space), 146
FormsSpaceFunctor (class in sage.modular.modform_hecketriangle.functors), 202
FormsSubSpaceFunctor (class in sage.modular.modform_hecketriangle.functors), 203
free_module() (sage.modular.local_comp.type_space.TypeSpace method), 361

G
G_inv() (sage.modular.modform_hecketriangle.abstract_ring.FormsRing_element.HeckeTriangleGroupElement method), 137
G_inv() (sage.modular.modform_hecketriangle.abstract_ring.FormsRing_element method), 273
galois_action() (sage.modular.dirichlet.DirichletCharacter method), 346
galois_action() (sage.modular.local_comp.type_space, 364
GaloisAction (sage.modular.local_comp.smoothchar, SmoothCharacterGeneric method), 302
GaloisAction (sage.modular.dirichlet.DirichletCharacter method), 318
Gamma_0_NFCusps() (in module sage.modular.cusps_nf), 429
gamma_array() (sage.modular.hypergeometric_motive.HypergeometricData method), 444
gamma_list() (sage.modular.hypergeometric_motive.HypergeometricData method), 454
gamma_list_to_cyclotomic() (in module sage.modular.hypergeometric_motive), 454
gauss_sum() (sage.modular.dirichlet.DirichletCharacter method), 302
gauss_sum_numerical() (sage.modular.dirichlet.DirichletCharacter method), 303
gauss_table() (sage.modular.hypergeometric_motive.HypergeometricData method), 444
gauss_table_full() (sage.modular.hypergeometric_motive.HypergeometricData method), 445
gen() (sage.modular.dirichlet.DirichletGroup_class method), 319
gen() (sage.modular.modform_ring.ModularFormsRing method), 89
gen() (sage.modular.modform_space.ModularFormsSpace method), 12
gen() (sage.modular.modform.hecketriangle.abstract_space.FormsSpace_abstract.method), 161

gens() (sage.modular.modform.element.GradedModularFormElement.method), 50

半integral_weight_modform_basis() (in module sage.modular.modform.half_integral), 85

half_product() (sage.modular.modform_hecketriangle.series_constructor.MFSeriesConstructor), 439

hecke_module_of_level() (in module sage.modular.modform.ambient_eps.ModularFormsAmbient), 391

hecke_module_of_level() (sage.modular.modform.element.Newform.method), 473

Index

496 Index
local_component() (sage.modular.modform.element.Newform method), 74
LocalComponent() (in module sage.modular.modform.element), 74
LocalComponentBase (class in sage.modular.local_comp.local_comp), 337
low_weight_bases() (in module sage.modular.overconvergent.hecke_series), 404
low_weight_generators() (in module sage.modular.overconvergent.hecke_series), 404
lseries() (sage.modular.modform.element.ModularFormAbstract method), 62
lseries() (sage.modular.modform_hecketriangle.element.FormsElement method), 172
Lvalue() (sage.modular.overconvergent.weightspace.AlgebraicWeight method), 377
Lvalue() (sage.modular.overconvergent.weightspace.WeightCharacter method), 379
M() (sage.modular.modform.element.EisensteinSeries method), 49
M() (sage.modular.quatalg.brandt.BrandtModule_class method), 420
M_value() (sage.modular.hypergeometric_motive.HypergeometricData method), 441
maximal_order() (in module sage.modular.quatalg.brandt), 427
maximal_order() (sage.modular.quatalg.brandt.BrandtModule_class method), 424
maximize_base_ring() (sage.modular.dirichlet.DirichletCharacter method), 309
merge() (sage.modular.modform.hecketriangle.functors.FormsRingFunctor method), 201
merge() (sage.modular.modform.hecketriangle.functors.FormsSpaceFunctor method), 202
merge() (sage.modular.modform.hecketriangle.functors.FormsSubSpaceFunctor method), 203
MeromorphicModularForms (class in sage.modular.modform.hecketriangle.space), 260
MeromorphicModularFormsRing (class in sage.modular.modform.hecketriangle.graded_ring), 260
MFSeriesConstructor (class in sage.modular.modform.hecketriangle.series_constructor), 270
minimal_twist() (sage.modular.local_comp.local_comp.ImprimitiveLocalComponent method), 336
minimal_twist() (sage.modular.local_comp.local_comp.PrimitiveLocalComponent method), 340
minimal_twist() (sage.modular.local_comp.type_space.TypeSpace method), 362
modular_forms_of_weight() (sage.modular.modform.ambient.ModularFormsAmbient method), 25
modular_forms_of_weight() (sage.modular.modform.ambient_eps.ModularFormsAmbient_eps method), 30
modular_forms_of_weight() (sage.modular.modform.ambient_R.ModularFormsAmbient_R method), 34
modular_forms_of_weight() (sage.modular.modform.cuspidal_submodule.CuspidalSubmodule method), 36
modular_forms_of_weight() (sage.modular.modform.element.Newform method), 74
modular_symbols() (sage.modular.modform_element.ModularFormElement method), 56
modular_symbols() (sage.modular.modform_element.Element method), 59
modular_symbols() (sage.modular.modform_element.Newform method), 75
modular_symbols() (sage.modular.modform.ambient.ModularFormsAmbient method), 81
modular_symbols() (sage.modular.modform.ambient_eps.ModularFormsAmbient_eps method), 82
modular_symbols() (sage.modular.modform.ambient_R.ModularFormsAmbient_R method), 83
modular_symbols() (sage.modular.quasimodform.ring.QuasiModularForms method), 283
modsym_eigenspace() (sage.modular.modform.element.Newform method), 75
S
S() (sage.modular.modform_hecketriangle.hecke_triangle_groups.HeckeTriangleGroup method), 204
sage.modular.buzzard
module, 335
sage.modular.cusps
module, 323
sage.modular.cusps_nf
module, 428
sage.modular.dims
module, 329
sage.modular.dirichlet
module, 295
sage.modular.etaproducts
module, 358
sage.modular.hypergeometric_motive
module, 438
sage.modular.local_comp.liftings
module, 365
sage.modular.local_comp.local_comp
module, 336
sage.modular.local_comp.smoothchar
module, 346
sage.modular.local_comp.type_space
module, 361
sage.modular.modform.ambient
module, 22
sage.modular.modform.ambient_eps
module, 28
sage.modular.modform.ambient_g0
module, 31
sage.modular.modform.ambient_g1
module, 32
sage.modular.modform.ambient_R
module, 33
sage.modular.modform.constructor
module, 1
sage.modular.modform.cuspidal_submodule
module, 35
sage.modular.modform.eis_series
module, 45
sage.modular.modform.eis_series_cython
module, 47
sage.modular.modform.eisenstein_submodule
module, 38
sage.modular.modform.element
module, 48
sage.modular.modform.half_integral
module, 85
sage.modular.modform.hecke_operator_on_qexp
module, 78
sage.modular.modform.j_invariant
module, 96
sage.modular.modform.notes
module, 98
sage.modular.modform.numerical
module, 79
sage.modular.modform.ring
module, 87
sage.modular.modform.space
module, 6
sage.modular.modform.submodule
module, 34
sage.modular.modform.theta
module, 97
sage.modular.modform.vm_basis
module, 83
sage.modular.modform.hecketriangle.abstract_ring
module, 120
sage.modular.modform.hecketriangle.abstract_space
module, 146
sage.modular.modform.hecketriangle.analytic_type
module, 253
sage.modular.modform.hecketriangle.constructor
module, 197
sage.modular.modform.hecketriangle.element
module, 171
sage.modular.modform.hecketriangle.functors
module, 201
sage.modular.modform.hecketriangle.graded_ring
module, 257
sage.modular.modform.hecketriangle.graded_ring_element
module, 174
sage.modular.modform.hecketriangle.hecke_triangle_group_element
module, 218
sage.modular.modform.hecketriangle.hecke_triangle_groups
module, 204
sage.modular.modform.hecketriangle.readme
module, 99
sage.modular.modform.hecketriangle.series_constructor
module, 270
sage.modular.modform.hecketriangle.space
module, 258
sage.modular.modform.hecketriangle.subspace
module, 266
sage.modular.modform.vm_basis
module, 83
sage.modular.overconvergent.genus0
module, 382
sage.modular.modform.vm_series
module, 395
sage.modular.overconvergent.hecke_series
module, 395
sage.modular.overconvergent.weightspace
module, 376
sage.modular.quasimodform.element
module, 286
sage.modular.quasimodform.vm_series
module, 277
sage.modular.quatalg.brandt
module, 417
sage.modular.ssmod.ssmod module, 407
sage_cusp() (sage.modular.etaproducts.CuspFamily method), 369
satake_polynomial() (sage.modular.local_comp.local_comp.UnramifiedPrincipalSeries method), 345
serre_derivative() (sage.modular.modform.element.GradedModuleElement method), 54
serre_derivative() (sage.modular.modform.element.ModularFormsElement method), 67
serre_derivative() (sage.modular.modform_hecketriangle.hecke_triangle_group_element.HeckeTriangleGroupElement method), 195
serre_derivative() (sage.modular.quasimodform.element.QuasiModularFormsElement method), 245
set_precision() (sage.modular.hypergeometric_motive.HypergeometricData method), 441
set_precision() (sage.modular.ssmod.ssmod method), 95
simple_elements() (sage.modular.modform.ring.ModularFormsRingElement method), 357
simplify() (sage.modular.multiple_zeta.Multizetas.Element method), 464
simplify() (sage.modular.multiple_zeta.Multizetas_iterated.Element method), 471
simplify_full() (sage.modular.multiple_zeta.Multizetas.Element method), 358
single_valued() (sage.modular.multiple_zeta.Multizetas.Element method), 360
span() (sage.modular.modform.space.ModularFormsSpace method), 341
span() (sage.modular.modform.space.ModularFormsSpace method), 18
span() (sage.modular.ssmod.ssmod method), 19
span_of_basis() (sage.modular.modform.space.ModularFormsSpace method), 19
species() (sage.modular.local_comp.local_comp.ImprimitiveLocalComponentBase method), 339
species() (sage.modular.local_comp.local_comp.LocalComponentBase method), 344
species() (sage.modular.local_comp.local_comp.PrincipalSeries method), 351
sqrt() (sage.modular.modform_hecketriangle.graded_ring_element.FormsRingElement method), 20
simple_elements() (sage.modular.modform_hecketriangle.hecke_future_group_element.HeckeTriangleGroupElement method), 413
string_repr() (sage.modular.modform_hecketriangle.hecke_triangle_group.HeckeTriangleGroup method), 19
sturm_bound() (in module sage.modular.dims), 334
sturm_bound() (sage.modular.modform.space.ModularFormsSpace method), 464
simple_fixed_point_set() (sage.modular.modform_hecketriangle.hecke_triangle_group_element.HeckeTriangleGroupElement method), 247
subgroup_gens() (sage.modular.ssmod.ssmod method), 351
subgroup_gens() (sage.modular.ssmod.ssmod method), 353
subgroup_gens() (sage.modular.ssmod.ssmod method), 358
subgroup_gens() (sage.modular.ssmod.ssmod method), 360
subspace() (sage.modular.modform_hecketriangle.abstract_space.FormsSpace method), 258
subspace() (sage.modular.modform_space.SubspaceForms method), 415
subspace() (sage.modular.modform_space.SubspaceForms method), 416
subspace() (sage.modular.modform_space.SubspaceForms method), 413
support() (sage.modular.modform.space.SupersingularModule method), 409
support() (sage.modular.modform.space.SupersingularModule method), 82
smooth_character_group_unramified_quadratic() (sage.modular.ssmod.ssmod method), 358
smooth_character_group_unramified_quadratic() (sage.modular.ssmod.ssmod method), 357
smooth_character_group_ramified_quadratic() (sage.modular.ssmod.ssmod method), 358
smooth_character_group_ramified_quadratic() (sage.modular.ssmod.ssmod method), 354
smooth_character_group_quad() (sage.modular.ssmod.ssmod method), 351
smooth_character_group_quad() (sage.modular.ssmod.ssmod method), 354
smooth_character_group_quad() (sage.modular.ssmod.ssmod method), 357
smooth_character_group_quad() (sage.modular.ssmod.ssmod method), 358
smooth_character_group_quad() (sage.modular.ssmod.ssmod method), 357
smooth_character_group_quad() (sage.modular.ssmod.ssmod method), 358
smooth_character_group_quad() (sage.modular.ssmod.ssmod method), 354
smooth_character_group_quad() (sage.modular.ssmod.ssmod method), 351
swap_alpha_beta() (sage.modular.hypergeometric_motive.HypergeometricData method), 449
symsquare_lseries() (sage.modular.modform.element.ModularForm_abstract method), 68
systems_of_abs() (sage.modular.modform.numerical.NumericalEigenforms method), 81
systems_of_eigenvalues() (sage.modular.modform.numerical.NumericalEigenforms method), 82
T
  t() (sage.modular.modform.element.EisensteinSeries method), 50
T() (sage.modular.modform_hecketriangle.hecke_triangle_groups.HeckeTriangleGroup
  HeckeTriangleGroup_class method), 205
  types() (sage.modular.dirichlet.DirichletGroup_class
  DirichletGroup_class method), 358
  unit_gens() (sage.modular.cusps_nf, 437
  units_on_gens() (sage.modular.modform.element.GradedModularFormElement
  GradedModularFormElement method), 311
theta2_qexp() (sage.modular.modform.theta
  theta2_qexp() method), 97
theta_qexp() (sage.modular.modform.theta
  theta_qexp() method), 97
to_polynomial() (sage.modular.modform.element.GradedModularForm_element
  GradedModularForm_element method), 449
to_polynomial() (sage.modular.modform_hecketriangle.hecke_triangle_groups.HeckeTriangleGroup
  HeckeTriangleGroup_class method), 205
trace() (sage.modular.hypergeometric_motive.HypergeometricData
  HypergeometricData method), 449
trace() (sage.modular.modform_hecketriangle.hecke_triangle_groups.HeckeTriangleGroup
  HeckeTriangleGroup_class method), 205
trivial_character() (in module sage.modular.dirichlet
  TrivialCharacter() method), 323
  TrivialCharacter() (in module sage.modular.dirichlet
  TrivialCharacter() method), 322
twist() (sage.modular.hypergeometric_motive.HypergeometricData
  HypergeometricData method), 450
twist() (sage.modular.modform.element.ModularFormElement
  ModularFormElement method), 57
twist() (sage.modular.modform.element.Newform
  Newform method), 75
twist_factor() (sage.modular.local_comp.local_comp.LocallyCompactCharacterGroup
  LocallyCompactCharacterGroup method), 339
twisting_character() (sage.modular.local_comp.local_comp.ImprimitiveLocallyCompactCharacterGroup
  LocallyCompactCharacterGroup method), 337
type_space() (sage.modular.local_comp.local_comp.PrimitiveSupercuspidalTypeSpace
  PrimitiveSupercuspidalTypeSpace method), 344
TypeSpace (class in sage.modular.local_comp.type_space
  TypeSpace class), 361
weight() (sage.modular.modform.element.ModularForm_dzero() method), 69
weight() (sage.modular.modform.numerical.NumericalEigenform.dzero() method), 82
weight() (sage.modular.modform.space.ModularFormsSpace.weight() method), 21
weight() (sage.modular.modform_hecketriangle.abstract_space.FormsSpace_abstract.weight() method), 170
weight() (sage.modular.modform_hecketriangle.graded_ring_element.FormsRingElement.weight() method), 196
weight() (sage.modular.overconvergent.genus0.OverconvergentModularFormElement.weight() method), 387
weight() (sage.modular.overconvergent.genus0.OverconvergentModularFormsSpace.weight() method), 394
weight() (sage.modular.quasimodform.element.QuasiModformsElement.weight() method), 293
weight() (sage.modular.ssmod.ssmod.SupersingularModule.weight() method), 414
weight_2_eisenstein_series() (sage.modular.quasimodform.ring.QuasiModularForms.weight_2_eisenstein_series() method), 285
weight_parameters() (sage.modular.modform_hecketriangle.abstract_space.FormsSpace_abstract.weight_parameters() method), 170
WeightCharacter (class in sage.modular.overconvergent.weightspace), 379
weights_list() (sage.modular.modform.element.GradedModularFormElement.weights_list() method), 55
weights_list() (sage.modular.quasimodform.element.QuasiModularFormsElement.weights_list() method), 293
WeightSpace_class (class in sage.modular.overconvergent.weightspace), 380
WeightSpace_constructor() (in module sage.modular.overconvergent.weightspace), 381
width() (sage.modular.etaproducts.CuspFamily.width() method), 369
wild_primes() (sage.modular.hypergeometric_motive.HypergeometricData.wild_primes() method), 451
word_S_T() (sage.modular.modform_hecketriangle.hecke_triangle_group_element.HeckeTriangleGroupElement.word_S_T() method), 251

Z
zero() (sage.modular.cusps_nf.NFCuspsSpace method), 435
zero() (sage.modular.modform.ring.ModularFormsRing.zero() method), 96
zero() (sage.modular.overconvergent.genus0.OverconvergentModularFormsSpace.zero() method), 395
zero() (sage.modular.overconvergent.weightspace.WeightSpace_class.zero() method), 381