# CONTENTS

## 1 Module List

1.1 Creating Spaces of Modular Forms ........................................... 1  
1.2 Generic spaces of modular forms ................................................ 6  
1.3 Ambient Spaces of Modular Forms .............................................. 21  
1.4 Modular Forms with Character .................................................. 26  
1.5 Modular Forms for \( \Gamma_0(N) \) over \( \mathbb{Q} \) .................... 29  
1.6 Modular Forms for \( \Gamma_1(N) \) and \( \Gamma_H(N) \) over \( \mathbb{Q} \) ...... 30  
1.7 Modular Forms over a Non-minimal Base Ring ............................ 31  
1.8 Submodules of spaces of modular forms .................................... 32  
1.9 The Cuspidal Subspace ........................................................... 33  
1.10 The Eisenstein Subspace ......................................................... 36  
1.11 Eisenstein Series ................................................................. 42  
1.12 Eisenstein Series (optimized compiled functions) ....................... 44  
1.13 Elements of modular forms spaces .......................................... 45  
1.14 Hecke Operators on \( q \)-expansions ........................................ 66  
1.15 Numerical computation of newforms ......................................... 68  
1.16 The Victor Miller Basis ......................................................... 71  
1.17 Compute spaces of half-integral weight modular forms ................. 73  
1.18 Graded Rings of Modular Forms .............................................. 75  
1.19 \( q \)-expansion of \( j \)-invariant .............................................. 81  
1.20 \( q \)-expansions of Theta Series ............................................ 81

## 2 Design Notes

2.1 Design Notes .............................................................................. 83

## 3 Indices and Tables

Python Module Index ........................................................................ 87

Index ................................................................................................... 89
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
```
Create an ambient space of modular forms.

**INPUT:**

- **group** - A congruence subgroup or a Dirichlet character $\epsilon$.
- **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 `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:

```python
sage: ModularForms(Gamma0(11), 2).dimension()
sage: ModularForms(Gamma0(1), 12).dimension()
```

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

```python
sage: ModularForms((1, 12)).dimension()
sage: ModularForms(11, 4)
```

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

```python
sage: ModularForms(Gamma1(13), 2)
```

We create a space with character:

```python
sage: e = (DirichletGroup(13).0)^2
sage: M = ModularForms(e, 2); M
sage: f = M.T(2).charpoly('x'); f
sage: f.factor()
```

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
Modular Forms space of dimension 20 for Congruence Subgroup Gamma_H(30) with H_generated by [11] of weight 2 over Rational Field
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 (trac ticket #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 trac ticket #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()
[]
sage: M == M.eisenstein_subspace()
True

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:

```python
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:

```python
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:

```python
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=None, 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:

```python
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=None, 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 $\text{Gal}(\overline{T}/F)$-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:**

```python
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:

```python
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

**OUTPUT:**

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

**EXAMPLES:**
.. code-block:: 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])
    (<type 'sage.rings.integer.Integer'>, 
     <class 'sage.modular.arithgroup.congroup_gamma0.Gamma0_class_with_category'>, 
     <type 'sage.rings.integer.Integer'>, 
     <type '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.

\[\text{sage.modular.modform.constructor.parse\_label}(s)\]

Given a string \(s\) corresponding to a newform label, return the corresponding group and index.

**EXAMPLES:**

\[\begin{align*}
\text{sage: & \quad \text{sage.modular.modform.constructor.parse\_label('11a')} \\
& \quad (\text{Congruence Subgroup Gamma0(11), 0}) \\
\text{sage: & \quad \text{sage.modular.modform.constructor.parse\_label('11aG1')} \\
& \quad (\text{Congruence Subgroup Gamma1(11), 0}) \\
\text{sage: & \quad \text{sage.modular.modform.constructor.parse\_label('11wG1')} \\
& \quad (\text{Congruence Subgroup Gamma1(11), 22})
\end{align*}\]

GammaH labels should also return the group and index (trac ticket #20823):

\[\begin{align*}
\text{sage: & \quad \text{sage.modular.modform.constructor.parse\_label('389cGH[16]')} \\
& \quad (\text{Congruence Subgroup Gamma\_H(389) with H generated by [16], 2})
\end{align*}\]

### 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 \(\mathbb{Q}\):

\[\begin{align*}
\text{sage: & \quad \text{ModularForms(11,2).base\_ring()} \\
& \quad \text{Rational Field} \\
\text{sage: & \quad \text{ModularForms(1,12).base\_ring()} \\
& \quad \text{Rational Field} \\
\text{sage: & \quad \text{CuspForms(Gamma1(13),3).base\_ring()} \\
& \quad \text{Rational Field}
\end{align*}\]

The base ring can be explicitly specified in the constructor function.

\[\begin{align*}
\text{sage: & \quad \text{ModularForms(11,2,base\_ring=GF(13)).base\_ring()} \\
& \quad \text{Finite Field of size 13}
\end{align*}\]

For modular forms with character the default base ring is the field generated by the image of the character.

\[\begin{align*}
\text{sage: & \quad \text{ModularForms(DirichletGroup(13).0,3).base\_ring()} \\
& \quad \text{Cyclotomic Field of order 12 and degree 4}
\end{align*}\]

For example, if the character is quadratic then the field is \(\mathbb{Q}\) (if the characteristic is 0).
An example in characteristic 7:

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

### class `sage.modular.modform.space.ModularFormsSpace`

```
Bases: `sage.modular.hecke.module.HeckeModule_generic`

A generic space of modular forms.
```

**Element**

alias of `sage.modular.modform.element.ModularFormElement`

**basis()**

Return a basis for self.

**EXAMPLES:**

```python
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) ]
```

**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:

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

Spaces of forms with nontrivial character:

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

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

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

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

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

If the level is one then the character is trivial.

### 1.2. Generic spaces of modular forms
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 trac ticket #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 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 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: 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: 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)]
sage: M = CuspForms(Gamma0(17),4, prec=10)
sage: M.echelon_basis()
[q + 2*q^5 - 8*q^7 - 8*q^8 + 7*q^9 + O(q^10),
q^2 - 3/2*q^5 - 7/2*q^6 + 9/2*q^7 + q^8 - 4*q^9 + O(q^10),
q^3 - 2*q^6 + q^7 - 4*q^8 - 2*q^9 + O(q^10),
q^4 - 1/2*q^5 - 5/2*q^6 + 3/2*q^7 + 2*q^9 + O(q^10)]
```

*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.

```
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()
```

(continues on next page)
We create a space with a funny basis then compute the corresponding echelon form.

```python
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),
  1 + O(q^6),
  q + 9*q^2 + 28*q^3 + 73*q^4 + 126*q^5 + O(q^6)]
```

```python
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)]
```

```python
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)]
```

eisenstein_series()  
Compute the Eisenstein series associated to this space.

**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.eisenstein_series()
Traceback (most recent call last):
```
... NotImplementedError: computation of Eisenstein series in this space not yet implemented

eisenstein_submodule()
Return the Eisenstein submodule for this space of modular forms.

EXAMPLES:

```
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
```

We check that a bug noticed on trac ticket #10450 is fixed:

```
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
```

eisenstein_subspace()
Synonym for eisenstein_submodule.

EXAMPLES:

```
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:

```
sage: N = ModularForms(6,4)
sage: N.dimension()
5
sage: N.embedded_submodule()
Vector space of dimension 5 over Rational Field
```

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.

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

EXAMPLES:

```
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
```

**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 + O(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 + O(q^6),
  q^3 + O(q^6) ]
```
\[ q^3 + O(q^6) \]

\textbf{group()}

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

**EXAMPLES:**

\begin{verbatim}
    sage: ModularForms(Gamma0(12),4).group()
    Congruence Subgroup Gamma0(12)

    sage: CuspForms(Gamma1(113),2).group()
    Congruence Subgroup Gamma1(113)
\end{verbatim}

Note that \( \Gamma_1(1) \) and \( \Gamma_0(1) \) are replaced by \( \text{SL}_2(\mathbb{Z}) \).

\begin{verbatim}
    sage: CuspForms(Gamma1(1),12).group()
    Modular Group SL(2,Z)

    sage: CuspForms(SL2Z,12).group()
    Modular Group SL(2,Z)
\end{verbatim}

\textbf{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.

\begin{verbatim}
    sage: CuspForms(Gamma0(11),2).has_character()
    True
\end{verbatim}

A space for \( \Gamma_1(N) \) (for \( N \geq 2 \)) never has a specific character.

\begin{verbatim}
    sage: CuspForms(Gamma1(11),2).has_character()
    False

    sage: CuspForms(DirichletGroup(11).0,3).has_character()
    True
\end{verbatim}

\textbf{integral_basis()}

Return an integral basis for this space of modular forms.

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

\begin{verbatim}
    sage: m = ModularForms(97,2,prec=10)
    sage: s = m cuspidal_subspace()
    sage: s.integral_basis()
    [q^3 + q^7 + 4*q^8 - 2*q^9 + O(q^10),
    q^2 + q^4 + q^7 + 3*q^8 - 3*q^9 + O(q^10),
    q^3 + q^4 - 3*q^8 + q^9 + O(q^10),
    2*q^4 - 2*q^8 + O(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^3 + q^7 + 4*q^8 - 2*q^9 + O(q^10),
    q^2 + q^4 + q^7 + 3*q^8 - 3*q^9 + O(q^10),
    q^3 + q^4 - 3*q^8 + q^9 + O(q^10),
    2*q^4 - 2*q^8 + O(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)
    ]
\end{verbatim}
Here’s another example where there is a big gap in the valuations:

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

\[ q + 2/3*q^9 + O(q^{10}),
q^2 + 2*q^8 - 5/3*q^9 + O(q^{10}),
q^3 - 2*q^8 + q^9 + O(q^{10}),
q^4 - q^8 + O(q^{10}),
q^5 - 2*q^8 + 2*q^9 + O(q^{10}),
q^6 + q^8 - 7/3*q^9 + O(q^{10}),
q^7 + 2*q^8 - 4/3*q^9 + O(q^{10})
\]

\[ q^2 + 2*q^8 - 5/3*q^9 + O(q^{10}),
q^3 - 2*q^8 + q^9 + O(q^{10}),
q^4 - q^8 + O(q^{10}),
q^5 - 2*q^8 + 2*q^9 + O(q^{10}),
q^6 + q^8 - 7/3*q^9 + O(q^{10}),
q^7 + 2*q^8 - 4/3*q^9 + O(q^{10})
\]
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:**

```sage
sage: M = sage.modular.modform.space.ModularFormsSpace(Gamma0(11), 2,
→DirichletGroup(1)[0], base_ring=QQ); M.modular_symbols()
Traceback (most recent call last):
... Not ImplementedError: computation of associated modular symbols space not yet implemented
```

**new_submodule** *(p=\texttt{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:**

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

**new_subspace** *(p=\texttt{None})*

Synonym for new_submodule.

**EXAMPLES:**

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

**newforms** *(\texttt{names=\texttt{None}})*

Return all newforms in the cuspidal subspace of self.

**EXAMPLES:**

```sage
sage: CuspForms(18,4).newforms()
[q + 2*q^2 + 4*q^4 - 6*q^5 + O(q^6)]
sage: CuspForms(32,4).newforms()
[q - 8*q^3 - 10*q^5 + O(q^6), q + 22*q^5 + O(q^6), q + 8*q^3 - 10*q^5 + O(q^6)]
sage: CuspForms(23).newforms('b')
[q + b0*q^2 + (-2*b0 - 1)*q^3 + (-b0 - 1)*q^4 + 2*b0*q^5 + O(q^6)]
sage: CuspForms(23).newforms()
```

(continues on next page)
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 (new\_prec=None)
Return or set the default precision used for displaying $q$-expansions of elements of this space.

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

OUTPUT: if 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()
[\( q - 24q^2 + 252q^3 - 1472q^4 + 4830q^5 + O(q^6) \)]
sage: S.prec(8)
8
sage: S.basis()
[\( q - 24q^2 + 252q^3 - 1472q^4 + 4830q^5 - 6048q^6 - 16744q^7 + O(q^8) \)]
```

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 - 2q^2 - q^3 + 2q^4 + q^5 + O(q^6), \]
  \( 1 + 12/5q + 36/5q^2 + 48/5q^3 + 84/5q^4 + 72/5q^5 + O(q^6) \)]

sage: M.q_echelon_basis()
[\( 1 + 12q^2 + 12q^3 + 12q^4 + 12q^5 + O(q^6), \]
  \( q - 2q^2 - q^3 + 2q^4 + q^5 + O(q^6) \)]
```

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 (>=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:

```python
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)]
```

```
q_integral_basis(prec=None)
Return a Z-reduced echelon basis of q-expansions for self.
The q-expansions are power series with coefficients in Z; they are not actual modular forms.
The base ring of self must be Q. The number of q-expansions returned equals the dimension.

EXAMPLES:

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

```
set_precision(new_prec)
Set the default precision used for displaying q-expansions.

INPUT:
new_prec - positive integer

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)
```

`span (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
```
**span_of_basis** \( (B) \)

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

**EXAMPLES:**

```python
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:**

```python
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
```

**REFERENCES:**

- [Stu1987]

**NOTE:**

1.2. Generic spaces of modular forms 19
Kevin Buzzard pointed out to me (William Stein) in Fall 2002 that the above bound is fine for \( \Gamma_0 \) 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 \( \mathbb{Z}[\varepsilon] \) 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:**

```python
sage: M = ModularForms(Gamma1(13),11)
sage: M.weight()
11
```

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

```python
sage: M = ModularForms(Gamma0(97),4)
sage: M.weight()
4
sage: M.eisenstein_submodule().weight()
4
```

**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:**

```python
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
```

**sage.modular.modform.space.is_ModularFormsSpace(\( x \))**
Return True if \( x \) is a `ModularFormsSpace`.

**EXAMPLES:**

```python
sage: from sage.modular.modform.space import is_ModularFormsSpace
sage: is_ModularFormsSpace(ModularForms(11,2))
True
sage: is_ModularFormsSpace(CuspForms(11,2))
True
sage: is_ModularFormsSpace(3)
False
```
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()
```

```python
[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 + 0(q^20), q^2 + 4*q^7 - q^8 + 8*q^12 + 2*q^13 + 10*q^17 - 5*q^18 + 0(q^20), q^3 + q^7 + 3*q^8 - q^12 + 5*q^13 + 3*q^17 + 6*q^18 + 0(q^20), q^4 - q^6 + 2*q^9 + 3*q^14 - 2*q^16 + 4*q^19 + 0(q^20), q^5 + q^10 + 2*q^15 + 0(q^20)]
```

```python
class sage.modular.modform.ambient.ModularFormsAmbient(group, weight, base_ring, character=None, eis_only=False)
Bases: sage.modular.modform.space.ModularFormsSpace, sage.modular.hecke.ambient_module.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:
```
```python
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:
```
• \( \mathbb{R} \)-ring

**EXAMPLES:**

```python
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),
l + 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.

```python
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),
l + q^3 + q^4 + 2*q^5 + O(q^6)
]
```

cuspidal_submodule()

Return the cuspidal submodule of this ambient module.

**EXAMPLES:**

```python
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:**

```python
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:**

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

Return all Eisenstein series associated to this space.

```
sage: ModularForms(27,2).eisenstein_series()
[ q^3 + O(q^6),
  q - 3*q^2 + 7*q^4 - 6*q^5 + O(q^6),
  1/12 + q + 3*q^2 + 7*q^4 + 6*q^5 + O(q^6),
  1/3 + q + 3*q^2 + 4*q^3 + 7*q^4 + 6*q^5 + O(q^6),
  13/12 + q + 3*q^2 + 4*q^3 + 7*q^4 + 6*q^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 - 15)*q^4 + q^5 + O(q^6),
  q + (zeta4 + 4)*q^2 + (-zeta4 + 9)*q^3 + (4*zeta4 + 15)*q^4 + 25*q^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 + 25*q^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 - 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) ]
```

eisenstein_submodule()

Return the Eisenstein submodule of this ambient module.

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(4) 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** \((n, \text{var}='x')\)
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** \((\text{sign}=0)\)
Return the corresponding space of modular symbols with the given sign.

**EXAMPLES:**

```
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:**
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:**

```sage
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:

```sage
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)]
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:

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

**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)]

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
```

`set_precision(n)`  
Set the default precision for displaying elements of this space.

EXAMPLES:

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

```sage
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
sage: M
Modular Forms space of dimension 66, character [zeta6, 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
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
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
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
```
class sage.modular.modform.ambient_eps.ModularFormsAmbient_eps(character, weight=2, base_ring=None, eis_only=False)

Bases: sage.modular.modform.ambient.ModularFormsAmbient

A space of modular forms with character.

change_ring(base_ring)

Return space with same defining parameters as this ambient space of modular symbols, but defined over a different base ring.

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.change_ring(CyclotomicField(12))
Modular Forms space of dimension 3, character [zeta6] and weight 2 over Cyclotomic Field of order 12 and degree 4
```

It must be possible to change the ring of the underlying Dirichlet character:

```sage
sage: m.change_ring(QQ)
Traceback (most recent call last):
...TypeError: Unable to coerce zeta6 to a rational
```

cuspidal submodule()

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

EXAMPLES:

```sage
es = DirichletGroup(4).0
sage: M = ModularForms(es, 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: M = ModularForms(eps^2, 2)
sage: M.modular_symbols()
Modular Symbols space of dimension 4 and level 13, weight 2, character \rightarrow [\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 \rightarrow [\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 \rightarrow [\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
Bases: sage.modular.modform.ambient.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 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 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:

```
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 + 72/5*q^4 + 144/5*q^5 + 192/5*q^6 + 96/5*q^7 + O(q^8)]
```

class `sage.modular.modform.ambient_g1.ModularFormsAmbient_g1_Q`

Bases: `sage.modular.modform.ambient_g1.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:

```
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:

```
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

```python
class sage.modular.modform.ambient_g1.ModularFormsAmbient_gH_Q(group, weight, eis_only)
Bases: sage.modular.modform.ambient.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: m = ModularForms(GammaH(100, [29]),2); m
Modular Forms space of dimension 48 for Congruence Subgroup Gamma_H(100) with
\( \Gamma_H \) 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
\( \Gamma_H \) = Congruence Subgroup Gamma_H(100) with \( \Gamma_H \) generated by [29] of weight 2 over
Rational Field
```

```python
eisenstein_submodule()
Return the Eisenstein submodule of this modular forms space.

EXAMPLES:
```
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 \( \Gamma_H \) generated by [29] of weight 3
over Rational Field
sage: type(E)
<class 'sage.modular.modform.eisenstein_submodule.EisensteinSubmodule_gH_Q_\nwith_category'>
```

### 1.7 Modular Forms over a Non-minimal Base Ring

```python
class sage.modular.modform.ambient_R.ModularFormsAmbient_R(M, base_ring)
Bases: sage.modular.modform.ambient.ModularFormsAmbient

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

EXAMPLES:
```
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
```

```python
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
```

cusp 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: `sage.modular.modform.space.ModularFormsSpace`, `sage.modular.hecke.submodule.HeckeSubmodule`

A submodule of an ambient space of modular forms.

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

Bases: `sage.modular.modform.submodule.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) ]

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) ]

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) ]
```

class sage.modular.modform.cuspidal_submodule.CuspidalSubmodule(ambient_space)
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)
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) of weight 12 with sign 0 over Rational Field

sage: eps = DirichletGroup(13).0
sage: S = CuspForms(eps^2, 2)
sage: 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 1 of 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: S.modular_symbols(sign=-1)
Modular Symbols subspace of dimension 1 of Modular Symbols space 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: sage.modular.modform.cuspidal_submodule.CuspidalSubmodule

Cuspidal submodule over a non-minimal base ring.

class sage.modular.modform.cuspidal_submodule.CuspidalSubmodule_eps(ambient_space)
Bases: sage.modular.modform.cuspidal_submodule.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
→[zeta4] 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: sage.modular.modform.cuspidal_submodule.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: sage.modular.modform.cuspidal_submodule.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: sage.modular.modform.cuspidal_submodule.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: sage.modular.modform.cuspidal_submodule.CuspidalSubmodule

Space of cusp forms of level 1 over \( \mathbb{Q} \).

class sage.modular.modform.cuspidal_submodule.CuspidalSubmodule_modsym_qexp(ambient_space)
Bases: sage.modular.modform.cuspidal_submodule.CuspidalSubmodule
cuspidal submodule with q-expansions calculated via modular symbols.

hecke_polynomial \( (n, var='x') \)
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:

```python
sage: CuspForms(105, 2).hecke_polynomial(2, '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^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^5 - 11*y^4 + 200*y^3 + 80*y^2
```

Check that trac ticket #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: sage.modular.modform.cuspidal_submodule.CuspidalSubmodule
Space of cusp forms of weight 1 with specified character.
class sage.modular.modform.cuspidal_submodule.CuspidalSubmodule_wt1_gH(ambient_space)
Bases: sage.modular.modform.cuspidal_submodule.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: sage.modular.modform.submodule.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:
sage: E = ModularForms(23,4).eisenstein_subspace()
sage: E == E.eisenstein_submodule()
True
```

```
modular_symbols(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:
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
\begin{verbatim}
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
\end{verbatim}

\begin{verbatim}
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(0)(1) of weight 12 with sign 0 over Rational Field
\end{verbatim}

\begin{verbatim}
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
\rightarrow 4 and level 13, weight 2, character [zeta6], sign 0, over Cyclotomic Field,
\rightarrow of order 6 and degree 2
\end{verbatim}

\begin{verbatim}
sage: E = EisensteinForms(eps, 1); E
Eisenstein subspace of dimension 1 of Modular Forms space of character
\rightarrow [zeta12] and weight 1 over Cyclotomic Field of order 12 and degree 4
\end{verbatim}

\begin{verbatim}
sage: E = EisensteinForms(eps^2, 2)
sage: E.modular_symbols()
Traceback (most recent call last):
... 
ValueError: the weight must be at least 2
\end{verbatim}

class sage.modular.modform.eisenstein_submodule.EisensteinSubmodule_eps(ambient_space)
Bases: sage.modular.modform.eisenstein_submodule.EisensteinSubmodule_params

Space of Eisenstein forms with given Dirichlet character.

EXAMPLES:

\begin{verbatim}
sage: e = DirichletGroup(27,CyclotomicField(3)).0**2
sage: M = ModularForms(e,2,prec=10).eisenstein_subspace()
sage: M.dimension()
6
\end{verbatim}

\begin{verbatim}
sage: M.eisenstein_series()
[ -1/3*zeta6 - 1/3 + q + (2*zeta6 - 1)*q^2 + q^3 + (-2*zeta6 - 1)*q^4 + (-5*zeta6 + 1)*q^5 + O(q^6),
-1/3*zeta6 - 1/3 + q^3 + O(q^6),
q + (-2*zeta6 + 1)*q^2 + (-2*zeta6 - 1)*q^4 + (5*zeta6 - 1)*q^5 + O(q^6),
q + (zeta6 + 1)*q^2 + 3*q^3 + (zeta6 + 2)*q^4 + (-zeta6 + 5)*q^5 + O(q^6),
q^3 + O(q^6),
q + (-zeta6 - 1)*q^2 + (zeta6 + 2)*q^4 + (zeta6 - 5)*q^5 + O(q^6),
]
\end{verbatim}

\begin{verbatim}
sage: M.eisenstein_subspace().T(2).matrix().fcp()
(x + 2*zeta3 + 1) * (x + zeta3 + 2) * (x - zeta3 - 2)^2 * (x - 2*zeta3 - 1)^2
\end{verbatim}

\begin{verbatim}
sage: ModularSymbols(e,2).eisenstein_subspace().T(2).matrix().fcp()
(x + 2*zeta3 + 1) * (x + zeta3 + 2) * (x - zeta3 - 2)^2 * (x - 2*zeta3 - 1)^2
\end{verbatim}

\begin{verbatim}
sage: M.basis()
[
]  
\end{verbatim}
1 - 3*zeta3*q^6 + (-2*zeta3 + 2)*q^9 + O(q^10),
q + (5*zeta3 + 5)*q^7 + O(q^10),
q^2 - 2*zeta3*q^8 + O(q^10),
q^3 + (zeta3 + 2)*q^6 + 3*q^9 + O(q^10),
q^4 - 2*zeta3*q^7 + O(q^10),
q^5 + (zeta3 + 1)*q^8 + O(q^10)
]

class sage.modular.modform.eisenstein_submodule.EisensteinSubmodule_g0_Q(ambient_space)
Bases: sage.modular.modform.eisenstein_submodule.EisensteinSubmodule_params
Space of Eisenstein forms for \Gamma_0(N).

class sage.modular.modform.eisenstein_submodule.EisensteinSubmodule_g1_Q(ambient_space)
Bases: sage.modular.modform.eisenstein_submodule.EisensteinSubmodule_gH_Q
Space of Eisenstein forms for \Gamma_1(N).

class sage.modular.modform.eisenstein_submodule.EisensteinSubmodule_gH_Q(ambient_space)
Bases: sage.modular.modform.eisenstein_submodule.EisensteinSubmodule_params
Space of Eisenstein forms for \Gamma_H(N).

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

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()
[1 + O(q^6),
q + 6*q^5 + O(q^6),
q^2 + O(q^6),
q^3 + O(q^6),
q^4 + O(q^6)
]
sage: E.change_ring(GF(5))
Eisenstein subspace of dimension 5 of Modular Forms space of dimension 5 for \Gamma_0(12) of weight 2 over Finite Field of size 5
sage: E.change_ring(GF(5)).basis()
[1 + O(q^6),
q + q^5 + O(q^6),
q^2 + O(q^6),
q^3 + O(q^6),
q^4 + O(q^6)
]

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

EXAMPLES:
sage: EisensteinForms(11,2).eisenstein_series()
[ 5/12 + q + 3*q^2 + 4*q^3 + 7*q^4 + 6*q^5 + O(q^6) ]
sage: EisensteinForms(1,4).eisenstein_series()
[ 1/240 + q + 9*q^2 + 28*q^3 + 73*q^4 + 126*q^5 + O(q^6) ]
sage: EisensteinForms(1,24).eisenstein_series()
[ 23634091/131040 + q + 8388609*q^2 + 94143178828*q^3 + 70368752566273*q^4 + \rightarrow 11920928955078126*q^5 + O(q^6) ]
sage: EisensteinForms(5,4).eisenstein_series()
[ 1/240 + q^5 + O(q^6) ]
sage: EisensteinForms(13,2).eisenstein_series()
[ 1/2 + q + 3*q^2 + 4*q^3 + 7*q^4 + 6*q^5 + O(q^6) ]
sage: E = EisensteinForms(Gamma1(7),2)
sage: E.set_precision(4)
sage: E.eisenstein_series()
[ 1/4 + q + 3*q^2 + 4*q^3 + O(q^4),
  1/7*zeta6 - 3/7 + q + (-2*zeta6 + 1)*q^2 + (3*zeta6 - 2)*q^3 + O(q^4),
  q + (-zeta6 + 3)*q^2 + (-zeta6 + 1)*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()
[
]
sage: M = ModularForms(GammaH(15, [4]), 4)
sage: M.eisenstein_series()
(continues on next page)
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),
1 + q^3 + O(q^6),
q + 7*q^2 + 26*q^3 + 57*q^4 + 125*q^5 + O(q^6),
q^3 + O(q^6)
]

def 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)]
```

def 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)
]
```

def parameters()
Return a list of parameters for each Eisenstein series spanning self. That is, for each such series, return a triple of the form (ψ, χ, level), where ψ and χ 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, Dirichlet character modulo 24 of conductor 1 mapping 7 |---> 1, 13 |---> 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 5 |---> 1, 11 |---> 1, 5 |---> 1, 11 |---> 1, 5 |---> 1, 12)
sage: pars = ModularForms(DirichletGroup(24).0,3).eisenstein_submodule().parameters()
sage: [(x[0].values_on_gens(),x[1].values_on_gens(),x[2]) for x in pars]
```

(continues on next page)
EisensteinForms(DirichletGroup(24).0,1).parameters()
[(Dirichlet character modulo 24 of conductor 1 mapping 7 |--> 1, 13 |--> 1, 17 |--> 1, Dirichlet character modulo 24 of conductor 4 mapping 7 |--> -1, 13 |--> 1, 17 |--> 1, 2), (Dirichlet character modulo 24 of conductor 1 mapping 7 |--> 1, 13 |--> 1, 17 |--> 1, Dirichlet character modulo 24 of conductor 4 mapping 7 |--> -1, 13 |--> 1, 17 |--> 1, 3), (Dirichlet character modulo 24 of conductor 1 mapping 7 |--> 1, 13 |--> 1, 17 |--> 1, Dirichlet character modulo 24 of conductor 4 mapping 7 |--> -1, 13 |--> 1, 17 |--> 1, 6)]

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 $M = \mathbb{Q}(\zeta_m)$):

**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:**

```python
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
```

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:

```python
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
```

### 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 `character` is an integer \(N\), then the parameters for \(\Gamma_1(N)\) are computed instead. Then the condition is that \(\chi(-1) \ast \psi(-1) = (-1)^k\).

If `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)]
```

(continues on next page)
sage.modular.modform.eis_series.compute_eisenstein_params(DirichletGroup(15).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, 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_asympt_coeffs=40)

Return the L-series of the weight \(2k\) Eisenstein series on \(SL_2(\mathbb{Z})\).

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_asympt_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)
-5.023553516459977977471968418348135050804419155747868718371029
```

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 \(SL_2(\mathbb{Z})\) to precision \(\text{prec}\) in the ring \(K\).

Three normalizations are available, depending on the parameter \(\text{normalization}\): 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(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 trac ticket #12043)

### 1.12 Eisenstein Series (optimized compiled functions)

\[ \text{sage.modular.modform.eis_series_cython.Ek_ZZ}(k, \text{prec}=10) \]

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:
• \( k – \text{int} \)
• \( \text{prec} – \text{int} \)

OUTPUT:
• list of Sage Integers.

EXAMPLES:

```
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 \( \text{prec} \) 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
    * ModularFormElement_elliptic_curve
    * EisensteinSeries

class sage.modular.modform.element.EisensteinSeries (parent, vector, t, chi, psi)

Bases: sage.modular.modform.element.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)
5/12 + q + 3*q^2 + 4*q^3 + 7*q^4 + 6*q^5 + O(q^6)
]
sage: E = EisensteinForms(Gamma1(7),2)
sage: E.set_precision(4)
sage: E.eisenstein_series()
[
1/4 + q + 3*q^2 + 4*q^3 + O(q^4),
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)
]

L() Return the conductor of self.chi().

EXAMPLES:

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

M() Return the conductor of self.psi().

EXAMPLES:

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

character() Return the character associated to self.

EXAMPLES:

sage: EisensteinForms(DirichletGroup(17).0,99).eisenstein_series()[1].character()
Dirichlet character modulo 17 of conductor 17 mapping 3 |--> zeta16
sage: chi = DirichletGroup(7)[4]
sage: E = EisensteinForms(chi).eisenstein_series() ; E
[
-1/7*zeta6 - 2/7 + q + (2*zeta6 - 1)*q^2 + (-3*zeta6 + 1)*q^3 + (-2*zeta6 - 1)*q^4 + (5*zeta6 - 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)
]
sage: E[0].character() == chi
True
sage: E[1].character() == chi
True

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], [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.ModularFormElement (parent, x, check=True)

Bases: sage.modular.modform.element.ModularForm_abstract, sage.modular.hecke.element.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 \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 – ignored (but accepted for compatibility with \Newform. atkin_lehner_eigenvalue())

OUTPUT:

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

See also:

For the conventions used to define the operator \( W_d \), see \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()
Traceback (most recent call last):
...
NotImplementedError: Don't know how to compute Atkin-Lehner matrix acting on this space (try using a newform constructor instead)

twist (chi, level=None)
Return the twist of the modular form \self by the Dirichlet character \chi.

If \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
- 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:
```python
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:

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

The base field of the twisted form is extended if necessary:

```python
sage: E4 = ModularForms(1, 4).gen(0)
sage: E4.parent()
Modular Forms space of dimension 1 for Modular Group SL(2,Z) of weight 4 over
→Rational Field
sage: chi = DirichletGroup(5)[1]
sage: chi.base_ring()
Cyclotomic Field of order 4 and degree 2
sage: E4_chi = E4.twist(chi)
sage: E4_chi.parent()
Modular Forms space of dimension 10, character [-1] and weight 4 over
→Cyclotomic Field of order 4 and degree 2
```

REFERENCES:

- [AL1978]
- [Kob1993]

AUTHORS:

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

class sage.modular.modform.element.ModularFormElement_elliptic_curve.

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

**atkin_lehner_eigenvalue**(d=None, embedding=None)

Return the result of the Atkin-Lehner operator $W_d$ on self.
INPUT:

• $d$ – a positive integer exactly dividing the level $N$ of self, i.e. $d$ divides $N$ and is coprime to $N/d$.
  (Defaults to $d = N$ if not given.)
• embedding – ignored (but accepted for compatibility with Newform.
  atkin_lehner_action())

OUTPUT:

The Atkin-Lehner eigenvalue of $W_d$ on self. This is either $1$ or $-1$.

EXAMPLES:

```python
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:

```python
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: sage.structure.element.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 this space (try using a newform constructor instead)

**base_ring()**

Return the base_ring of self.

**EXAMPLES:**

```python
sage: (ModularForms(117, 2).13).base_ring()
Rational Field
sage: (ModularForms(119, 2, base_ring=GF(7)).12).base_ring()
Finite Field of size 7
```

**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:**

```python
sage: ModularForms(DirichletGroup(17).0^2, 2).2.character()
Dirichlet character modulo 17 of conductor 17 mapping 3 |--> zeta8
sage: CuspForms(Gamma1(7), 3).gen(0).character()  # compute=False
Traceback (most recent call last):
  ...
ValueError: Form is not an eigenvector for <3>
```

**cm_discriminant()**

Return the discriminant of the CM field associated to this form. An error will be raised if the form isn’t of CM type.

**EXAMPLES:**

```python
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
```

**coefficients**(X)

The coefficients a_n of self, for integers n>=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.

**group()**

Return the group for which self is a modular form.
EXAMPLES:

```python
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:

```python
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.

```python
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.

```python
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.

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

```python
sage: abs(L.check_functional_equation()) < 1.0e-20
True
```

For non-rational newforms we can specify an embedding of the coefficient field:

```python
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 (\( \mathbb{Q}(\zeta_3) \) in this case):
We compute with the L-series of the Eisenstein series $E_4$:

```python
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):

```python
sage: delta = CuspForms(1,12).0
sage: L = delta.lseries()
sage: L(1)
0.0374412812685155
sage: L = delta_lseries()
sage: L(1)
0.0374412812685155
```

We check that trac ticket #5262 is fixed:

```python
sage: E = EllipticCurve('37b2')
sage: h = Newforms(37)[1]
sage: LE = E.lseries()
sage: Lh = h.lseries()
sage: Lh(1), LE(1)
(0.725681061936153, 0.725681061936153)
```

We check that trac ticket #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)
sage: L5 = f5.lseries()
sage: abs(L5.check_functional_equation()) < 1e-15
True
sage: abs(L5(4) - (gamma(1/4)^8/(3840*pi^2)).n()) < 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
sage: L = f.lseries(prec=53)
sage: abs(L(1)) < 2^-53
True
sage: L = f.lseries(prec=100)
```

(continues on next page)
sage: abs(L(1)) < 2^-100
True
sage: f = Newforms(27, names='a')[0]
sage: L = f.lseries()
sage: L(1)
0.588879583428483

**padded_list**(n)

Return a list of length n whose entries are the first n coefficients of the q-expansion of self.

**EXAMPLES:**

```
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]
```

**period**(M, prec=53)

Return the period of self with respect to M.

**INPUT:**

- `self` – a cusp form f of weight 2 for Gamma0(N)
- `M` – an element of Γ₀(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 α in P¹(Q):

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

This is independent of the choice of α.

**EXAMPLES:**

```
sage: C = Newforms(11, 2)[0]
sage: m = C.group()(matrix([[-4, -3], [11, 8]]))
sage: C.period(m)
-0.634604652139777 - 1.45881661693850*I

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

If E is an elliptic curve over 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:

```
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
```

(continues on next page)
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 \texttt{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 $E$:

\begin{verbatim}
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
\end{verbatim}

\textbf{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 \texttt{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

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

\textbf{INPUT:}

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

\textbf{EXAMPLES:}

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

The Petersson norm depends on a choice of embedding:

\begin{verbatim}
sage: set_verbose(-2, "dokchitser.py") # disable precision-loss warnings
sage: F = Newforms(1, 24, names='a')[0]
sage: F.petersson_norm(embedding=0)
\end{verbatim}
prec()
Return the precision to which self.q_expansion() is currently known. Note that this may be 0.

EXAMPLES:

```python
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()
20
```

q_expansion(prec=None)
The \(q\)-expansion of the modular form to precision \(O(q^{prec})\). This function takes one argument, which is the integer prec.

EXAMPLES:

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

```python
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:

```python
sage: f = CuspForms(23,2).0
sage: f.q_expansion()
q + O(q^2)
sage: f.q_expansion(1)
O(q^1)
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
```

qexp(prec=None)
Same as self.q_expansion(prec).

See also:

q_expansion()
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:**

An example twisted by a nontrivial character:

```python
sage: psi = DirichletGroup(7).0^2
sage: L = CuspForms(1, 16).0.symsquare_lseries(psi)
sage: L(22)
0.998407750967420 - 0.00295712911510708*I
```

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)
sage: L(5)
verbose -1 (...: dokchitser.py, __call__) Warning: Loss of 8 decimal digits due to cancellation
-3.57698266793901e19
```

**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:**

```python
sage: ModularForms(11,2).0.valuation()
1
sage: ModularForms(11,2).1.valuation()
0
sage: ModularForms(25,6).1.valuation()
2
sage: ModularForms(25,6).6.valuation()
7
```
weight()
Return the weight of self.

EXAMPLES:

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

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

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()  # most recent call last):
...  # traceback
    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

\[
f | W_d = \lambda_d(f)f'.
\]

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):
    sage: (continues on next page)
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))
(1.00000000000000, 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))
(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))
(-0.894427190999916 + 0.447213595499958*I, q + a0*q^2 + a0*q^3 - q^4 + (a0 - 2)*q^5 + O(q^6))
```

We can compute the eigenvalue of $W_p$ in certain cases where the $p$-th coefficient of $f$ is zero:

```
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:

```
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

.. math::
   f \mid W_d = \lambda_d(f) f'

where $f'$ is some normalised newform (not necessarily equal to $f$).

If `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 `normalisation='arithmetic'` we compute instead the quotient

.. math::
   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.)

**See also:**

- `sage.modular.hecke.module.atkin_lehner_operator()` (especially for the conventions used to define the operator $W_d$)

- `atkin_lehner_action()`, which returns both the pseudo-eigenvalue and the newform $f'$.

**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)
```

(continues on next page)
Traceback (most recent call last):
...
ValueError: Unable to compute square root. Try specifying an embedding into a
→ larger ring
sage: L = f.hecke_eigenvalue_field(); x = polygen(QQ); M.<sqrt5> = L.
→ extension(x^2 - 5)
 1/5*a2*sqrt5
sage: f.atkin_lehner_eigenvalue(5, embedding=M.coerce_map_from(L))
1/5*a2*sqrt5
sage: f.atkin_lehner_eigenvalue(5, normalization='arithmetic')
a2
sage: Newforms(DirichletGroup(5).0^2, 6, names='a')[0].atkin_lehner_→eigenvalue()
Traceback (most recent call last):
...
ValueError: Unable to compute Gauss sum. Try specifying an embedding into a
→ larger ring

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

EXAMPLES:

sage: Newforms(Gamma1(7), 4, names='a')[1].character()
Dirichlet character modulo 7 of conductor 7 mapping 3 |--> 1/2*a1
sage: chi = DirichletGroup(3).0; Newforms(chi, 7)[0].character() == chi
True

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 + 2*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

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
EXAMPLES:

```python
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: ls[2][3].minpoly()
x^2 - 9*x + 2
```

**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),
qu + 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
```

**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:

```python
sage: N = Newform("37a")
sage: N.modular_symbols(0)
Modular Symbols subspace of dimension 2 of Modular Symbols space of dimension 6 for Gamma_0(37) of weight 2 with sign 0 over Rational Field
```

1.13. Elements of modular forms spaces
sage: M = N.modular_symbols(0)
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\( (\text{sign}=0)\)

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

EXAMPLES:

```python
sage: f = Newforms(18, 4)[0]
sage: f.modular_symbols()
Modular Symbols subspace of dimension 2 of Modular Symbols space of dimension \to 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 \to 11 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:

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

twist\( (\text{chi, level=None, check=True})\)

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. By default, the algorithm tries to compute \(N\) using [AL1978], Theorem 3.1.
• 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 that a wrong result is returned if an incorrect 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)
```

AUTHORS:

• Peter Bruin (April 2015)

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:

1.13. Elements of modular forms spaces 65
sage: L = delta_lseries()
sage: L(1)
0.0374412812685155

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

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

Return True if x is a modular form.

EXAMPLES:

```python
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

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 >= 1
- \( k \) - an integer
- \( \varepsilon \) - Dirichlet character
- `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()
[ q - 24*q^2 + 252*q^3 - 1472*q^4 + 4830*q^5 + O(q^6),
  1 + 65520/691*q + 134250480/691*q^2 + 11606736960/691*q^3 + 274945048560/691*q^4
    + 3199218815520/691*q^5 + O(q^6)
]
sage: hecke_operator_on_basis(ModularForms(1,12).q_expansion_basis(), 3, 12)
```

(continues on next page)
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]
```

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:**

```
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)
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], 1, 12)
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: (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[[q]]
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 $p$: 1.14. Hecke Operators on $q$-expansions
1.15 Numerical computation of newforms

```
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
```

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

Bases: `sage.structure.sage_object.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
- `weight` - an integer >= 2
- `eps` - a small float; abs( ) < eps is what “equal to zero” is interpreted as  for floating point numbers.
- `delta` - a small-ish float; eigenvalues are considered distinct if their difference has absolute value at least delta
- `tp` - use the Hecke operators T_p for p in tp when searching for a random Hecke operator with distinct Hecke eigenvalues.

**OUTPUT:**

A numerical eigenforms object, with the following useful methods:

- `ap()` - return all eigenvalues of T_p
- `eigenvalues()` - list of eigenvalues corresponding to the given list of primes, e.g.,:

  ```
  [[eigenvalues of T_2],
   [eigenvalues of T_3],
   [eigenvalues of T_5], ...]
  ```

- `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.

**EXAMPLES:**

```
sage: n = numerical_eigenforms(23)
sage: n == loads(dumps(n))
True
sage: n.ap(2)  # rel tol 2e-14
[3.0, -1.6180339887498947, 0.6180339887498968]
sage: n.systems_of_eigenvalues(7)  # rel tol 2e-14
[[-1.6180339887498947, 2.2360679774997894, -3.2360679774997894],
 [0.6180339887498968, -2.236067977499788, 1.2360679774997936],
```

(continues on next page)
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.

**INPUT:**

- p - integer, a prime number

**OUTPUT:**

- 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 \( \text{prime} \)

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])  # rel tol 2.4e-10
[[177148.0, 252.
  00000000001896], [48828126.0, 4830.
  0000000001376], [-1792160394038.0, -577737.9999898539]]
```

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 6e-14
[0.3111078174659775, 2.903211925911551, 2.525427560843529, 3.214319743377552],
[1.0, 2.000000000000027, 3.00000000000003, 1.000000000000044],
[1.4811943040920152, 0.8060634335253695, 3.1563251746586642, 0.
  →675130870566477],
[2.17008486262034, 1.7092753594369208, 1.63089761381512, 0.
  →46081112718908984],
[3.0, 4.0, 6.0, 8.0]
```

**systems_of_eigenvalues(bound)**

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

EXAMPLES:

```python
sage: numerical_eigenforms(61).systems_of_eigenvalues(10) # rel tol 6e-14
[-1.4811943040920152, 0.8060634335253695, 3.1563251746586642, 0.
  →675130870566477],
[-1.0, -2.000000000000027, -3.00000000000003, 1.000000000000044],
[0.3111078174659775, 2.903211925911551, -2.525427560843529, -3.
  →214319743377552],
[2.17008486262034, -1.7092753594369208, -1.63089761381512, -0.
  →46081112718908984],
[3.0, 4.0, 6.0, 8.0]
```

**weight()**

Return the weight of this set of modular eigenforms.

EXAMPLES:

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

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

Given a vector \( v \) and a threshold \( \varepsilon \), return all indices where \( |v| \) is larger than \( \varepsilon \).

EXAMPLES:
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: 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
```

EXAMPLES:

```
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'))
```

(continues on next page)
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)

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) 
]
sage: victor_miller_basis(24, 6)
[ 1 + 52416000*q^3 + 39007332000*q^4 + 6609020221440*q^5 + O(q^6),
  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) 
]
sage: victor_miller_basis(32, 6)
[ 1 + 2611200*q^3 + 19524758400*q^4 + 19715347537920*q^5 + O(q^6),
  q + 50220*q^3 + 87866368*q^4 + 18647219790*q^5 + O(q^6),
  q^2 + 432*q^3 + 39960*q^4 - 1418560*q^5 + O(q^6) 
]
sage: victor_miller_basis(40,200)[1:] == victor_miller_basis(40,200,cusp_only=True)
True
sage: victor_miller_basis(200,40)[1:] == victor_miller_basis(200,40,cusp_only=True)
True

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 Basmaji’s thesis.

AUTHORS:

- William Stein (2007-08)

```python
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:

- `chi` – a Dirichlet character with modulus divisible by 16
- `k` – an odd integer $> 1$
- `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 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)
```

```plaintext
[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)
```

```plaintext
[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)
```

```plaintext
[]
```

```sage
half_integral_weight_modform_basis(DirichletGroup(16,QQ).1, 5, 10)
```

```plaintext
[q - 2*q^3 - 2*q^5 + 4*q^7 - q^9 + O(q^10)]
```

```sage
half_integral_weight_modform_basis(DirichletGroup(16,QQ).1, 7, 10)
```

```plaintext
[q - 2*q^2 + 4*q^3 + 4*q^4 - 10*q^5 + 19*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)]
```

```sage
half_integral_weight_modform_basis(DirichletGroup(16,QQ).1, 9, 10)
```

```plaintext
[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^3 + 4*q^4 - 8*q^5 - 8*q^6 + 20*q^7 - 8*q^8 + 32*q^9 + O(q^10),
 q^3 - 2*q^4 + 4*q^5 + 4*q^6 - 10*q^7 - 16*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 trac ticket #5792).

```sage
half_integral_weight_modform_basis(trivial_character(16),9,10)
```

```plaintext
[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 + O(q^10)]
```


Let $S = S_{k+1}(\varepsilon)$ be the space of cusp forms of even integer weight $k + 1$ and character $\varepsilon = \chi\psi^{(k+1)/2}$, 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:


```python
class sage.modular.modform.find_generators.ModularFormsRing(group, base_ring=RationalField)
```

Bases: `sage.structure.sage_object.SageObject`

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

INPUT:

- `group` – a congruence subgroup of \( SL_2(\mathbb{Z}) \), or a positive integer \( N \) (interpreted as \( \Gamma_0(N) \))
- `base_ring` (ring, default: \( \mathbb{Q} \)) – a base ring, which should be \( \mathbb{Q}, \mathbb{Z}, \) or the integers mod \( p \) for some prime \( p \).

EXAMPLES:

```python
sage: ModularFormsRing(Gamma1(13))
Ring of modular forms for Congruence Subgroup Gamma1(13) with coefficients in Rational Field
sage: m = ModularFormsRing(4); m
Ring of modular forms for Congruence Subgroup Gamma0(4) with coefficients in 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))]
```

`base_ring()`

Return the coefficient ring of this modular forms ring.

EXAMPLES:
sage: ModularFormsRing(Gamma1(13)).base_ring()
Rational Field
sage: ModularFormsRing(Gamma1(13), base_ring = ZZ).base_ring()
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, 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 + O(q^8), q^3 + 456*q^6 + O(q^8), q^4 - 135*q^6 - 926*q^7 + O(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.
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

gen_forms (maxweight=8, start_gens=[], start_weight=2)
This function calculates a list of modular forms generating this ring (as an algebra over the appropriate base ring). It differs from generators() only in that it returns Sage modular form objects, rather than bare $q$-expansions; and if the base ring is a finite field, the modular forms returned will be forms in characteristic 0 with integral $q$-expansions whose reductions modulo $p$ generate the ring of modular forms mod $p$.

INPUT:

- maxweight (integer, default: 8) – calculate forms generating all forms up to this weight.
• `start_gens` (list, default: `[]`) – a list of modular forms. If this list is nonempty, we find a minimal generating set containing these forms.

• `start_weight` (integer, default: 2) – calculate the graded subalgebra of forms of weight at least `start_weight`.

**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 `generators()`). If called with non-default values for these parameters, caching will be disabled.

**EXAMPLES:**

```sage
c = ModularFormsRing(Gamma0(11), Zmod(5)).gen_forms(); c
[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)]
c[0].parent()
Modular Forms space of dimension 2 for Congruence Subgroup Gamma0(11) of weight 2 over Rational Field
```
sage: ModularFormsRing(SL2Z).generators()
[(4, 1 + 240*q + 2160*q^2 + 6720*q^3 + 17520*q^4 + 30240*q^5 + 60480*q^6 + 82560*q^7 + 140400*q^8 + 181680*q^9 + O(q^10)),
 (6, 1 - 504*q - 16632*q^2 - 122976*q^3 - 532728*q^4 - 1575504*q^5 - 4058208*q^6 - 8471232*q^7 - 17047800*q^8 - 29883672*q^9 + O(q^10))]
sage: s = ModularFormsRing(SL2Z).generators(maxweight=5, prec=3); s
[(4, 1 + 240*q + 2160*q^2 + O(q^3))]
sage: s[0][1].parent()
Power Series Ring in q over Rational Field
sage: ModularFormsRing(1).generators(prec=4)
[(4, 1 + 240*q^2 + 2160*q^4 + O(q^4)), (6, 1 - 504*q - 16632*q^2 - 122976*q^3 + O(q^4))]
sage: ModularFormsRing(2).generators(prec=12)
[(2, 1 + 24*q + 24*q^2 + 9673q^4 + 144q^5 + 96*q^6 + 192q^7 + 24q^8 + 18q^9 + O(q^10)),
 (2, 1 - 24q + 24q^2 + 9673q^4 + 144q^5 + 96*q^6 + 192q^7 + 24q^8 + 18q^9 + O(q^10)),
 (3, 1 + 12q^2 + 64q^3 + 60q^4 + 160q^6 + 384q^7 + 252q^8 + O(q^10)), (3, q + 4q^3 + 6q^5 + 8q^7 + 32q^9 + O(q^10))]
sage: ModularFormsRing(4).generators(maxweight=2, prec=20)
[(2, 1 + 24*q + 24*q^2 + 24q^3 + 96*q^4 + 144q^5 + 96*q^6 + 192q^7 + 24q^8 + 18q^9 + O(q^10)),
 (2, 1 - 24q + 24q^2 + 24q^3 + 9673q^4 + 144q^5 + 96*q^6 + 192q^7 + 24q^8 + 18q^9 + O(q^10)),
 (3, q + 4q^3 + 6q^5 + 8q^7 + 32q^9 + 64q^10 + 96q^11 + 128q^12 + 192q^13 + 256q^14 + 384q^15 + 512q^16 + O(q^20)),
 (3, q + 4q^3 + 6q^5 + 8q^7 + 32q^9 + 64q^10 + 96q^11 + 128q^12 + 192q^13 + 256q^14 + 384q^15 + 512q^16 + O(q^20))]

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: dimension_modular_forms(11, 2)
2
sage: dimension_modular_forms(11, 4)
4
```

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 \(\geq 2\):

```
sage: ModularFormsRing(Gamma1(4)).generators(prec=10, maxweight=10)
[(2, 1 + 24*q^2 + 24q^4 + 96*q^6 + 24q^8 + O(q^10)),
 (2, q + 4q^3 + 6q^5 + 8q^7 + 32q^9 + 64q^10 + 96q^11 + 128q^12 + 192q^13 + 256q^14 + 384q^15 + 512q^16 + O(q^20)),
 (3, q + 4q^3 + 6q^5 + 8q^7 + 32q^9 + 64q^10 + 96q^11 + 128q^12 + 192q^13 + 256q^14 + 384q^15 + 512q^16 + O(q^20))]
```

Using different base rings will change the generators:

```
sage: ModularFormsRing(Gamma0(13)).generators(maxweight=12, prec=4)
[(2, 1 + 2q + 6q^2 + 8q^3 + O(q^4)), (4, q + O(q^4)), (4, q^2 + O(q^4)), (4, q^3 + O(q^4)), (6, 1 + O(q^4)), (6, q + O(q^4))]
sage: ModularFormsRing(Gamma0(13), base_ring=ZZ).generators(maxweight=12, prec=4)
[(2, 1 + 2q + 6q^2 + 8q^3 + O(q^4)), (4, q + O(q^4)), (4, q^2 + O(q^4)), (4, q^3 + O(q^4)), (6, 0 + O(q^4)), (6, O(q^4)), (12, O(q^4))]
sage: [k for k, f in ModularFormsRing(1, QQ).generators(maxweight=12)]
[4, 6]
```

(continues on next page)
An example where `start_gens` are specified:

```python
sage: M = ModularForms(11, 2); f = (M.0 + M.1).qexp(8)
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!
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)
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 + O(q^10)), (4, 1 + O(q^10))]
```

group()

Return the congruence subgroup for which this is the ring of modular forms.

EXAMPLES:

```python
sage: R = ModularFormsRing(Gamma1(13))
sage: R.group() is Gamma1(13)
True
```

`modular_forms_of_weight` *(weight)*

Return the space of modular forms on this group of the given weight.

EXAMPLES:

```python
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
```

`q_expansion_basis` *(weight, prec=None, 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 `find_generators`.

INPUT:

- `weight` (integer) – the weight
- `prec` (integer or None, default: None) – power series precision. If None, the precision defaults to the Sturm bound for the requested level and weight.
- `use_random` (boolean, default: 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.
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 +
 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
sage: A == B  # long time
True
```

Check that absurdly small values of `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), O(q^5),
 O(q^5), O(q^5), O(q^5), O(q^5)]
```

```
sage.modular.modform.find_generators.basis_for_modform_space(*args)
This function, which existed in earlier versions of Sage, has now been replaced by the `q_expansion_basis()` method of ModularFormsRing objects.

EXAMPLES:

```
sage: from sage.modular.modform.find_generators import basis_for_modform_space
sage: basis_for_modform_space()
Traceback (most recent call last):
  ...
NotImplementedError: basis_for_modform_space has been removed -- use
  ModularFormsRing.q_expansion_basis()
```

```
sage.modular.modform.find_generators.find_generators(*args)
This function, which existed in earlier versions of Sage, has now been replaced by the `generators()` method of ModularFormsRing objects.

EXAMPLES:

```
sage: from sage.modular.modform.find_generators import find_generators
sage: find_generators()
Traceback (most recent call last):
  ...
NotImplementedError: find_generators has been removed -- use ModularFormsRing.
  generators()
```
1.19 q-expansion of j-invariant

```python
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:**

```python
sage: j_invariant_qexp(4)
q^-1 + 744 + 196884*q + 21493760*q^2 + 864299970*q^3 + O(q^4)
```

1.20 q-expansions of Theta Series

**AUTHOR:**

William Stein

```python
sage.modular.modform.theta.theta2_qexp(prec=10, var='q', K=Integer Ring, sparse=False)
```

Return the $q$-expansion of the series \( \theta_2 = \sum_{n \text{ odd}} q^{n^2}. \)

**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:**

```python
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
```
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 \( \theta = 1 + 2\sum_{n=1}^{\infty} q^{n^2}. \)

**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)
1 + 2*q + 2*q^4 + 2*q^9 + O(q^10)
sage: theta_qexp(100)
1 + 2*q + 2*q^4 + 2*q^9 + 2*q^16 + 2*q^25 + 2*q^36 + 2*q^49 + 2*q^64 + 2*q^81 + <\ldots> +O(q^100)
sage: theta_qexp(100, 't')
1 + 2*t + 2*t^4 + 2*t^9 + 2*t^16 + 2*t^25 + 2*t^36 + 2*t^49 + 2*t^64 + 2*t^81 + <\ldots> +O(t^100)
sage: theta_qexp(100, 't', GF(2))
1 + 0*t^100
sage: f = theta_qexp(20, sparse=True); f
1 + 2*q + 2*q^4 + 2*q^9 + 2*q^16 + O(q^20)
sage: parent(f)
Sparse Power Series Ring in q over Integer Ring
```
2.1 Design Notes

The implementation depends 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 submodulesq 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 ZZ, which is confusing.
CHAPTER THREE

INDICES AND TABLES

• Index
• Module Index
• Search Page
m
sage.modular.modform.ambient, 21
sage.modular.modform.ambient_eps, 26
sage.modular.modform.ambient_g0, 29
sage.modular.modform.ambient_g1, 30
sage.modular.modform.ambient_R, 31
sage.modular.modform.constructor, 1
sage.modular.modform.cuspidal_submodule, 33
sage.modular.modform.eis_series, 42
sage.modular.modform.eis_series_cython, 44
sage.modular.modform.eisenstein_submodule, 36
sage.modular.modform.element, 45
sage.modular.modform.find_generators, 75
sage.modular.modform.half_integral, 73
sage.modular.modform.hecke_operator_on_qexp, 66
sage.modular.modform.j_invariant, 81
sage.modular.modform.notes, 83
sage.modular.modform.numerical, 68
sage.modular.modform.space, 6
sage.modular.modform.submodule, 32
sage.modular.modform.theta, 81
sage.modular.modform.vm_basis, 71
INDEX

A

abelian_variety() (sage.modular.modform.element.Newform method), 59
ambient_space() (sage.modular.modform.ambient.ModularFormsAmbient method), 21
ap() (sage.modular.modform.numerical.NumericalEigenforms method), 69
atkin_lehner_action() (sage.modular.modform.element.Newform method), 59
atkin_lehner_eigenvalue() (sage.modular.modform.element.ModularForm_abstract method), 50
atkin_lehner_eigenvalue() (sage.modular.modform.element.ModularFormElement method), 48
atkin_lehner_eigenvalue() (sage.modular.modform.element.ModularFormElement_elliptic_curve method), 49
atkin_lehner_eigenvalue() (sage.modular.modform.element.Newform method), 60

B

base_ring() (sage.modular.modform.element.ModularForm_abstract method), 51
base_ring() (sage.modular.modform.find_generators.ModularFormsRing method), 75
basis() (sage.modular.modform.space.ModularFormsSpace method), 7
basis_for_modform_space() (in module sage.modular.modform.find_generators), 80

C

canonical_parameters() (in module sage.modular.modform.constructor), 5
change_ring() (sage.modular.modform.ambient.ModularFormsAmbient method), 21
change_ring() (sage.modular.modform.ambient_eps.ModularFormsAmbient_eps method), 28
change_ring() (sage.modular.modform.ambient_R.ModularFormsAmbient_R method), 31
change_ring() (sage.modular.modform.cuspidal_submodule.CuspidalSubmodule method), 33
change_ring() (sage.modular.modform.eisenstein_submodule.EisensteinSubmodule_params method), 38
character() (sage.modular.modform.element.EisensteinSeries method), 46
character() (sage.modular.modform.element.ModularForm_abstract method), 51
character() (sage.modular.modform.element.Newform method), 62
character() (sage.modular.modform.space.ModularFormsSpace method), 7
chi() (sage.modular.modform.element.EisensteinSeries method), 46
cm_discriminant() (sage.modular.modform.element.ModularForm_abstract method), 51
coefficient() (sage.modular.modform.element.Newform method), 62
coefficients() (sage.modular.modform.element.ModularForm_abstract method), 51
compute_eisenstein_params() (in module sage.modular.modform.eis_series), 42
contains_each() (in module sage.modular.modform.space), 20
CuspForms() (in module sage.modular.modform.constructor), 1
cuspidal_ideal_generators() (sage.modular.modform.find_generators.ModularFormsRing method), 76
cuspidal_submodule() (sage.modular.modform.ambient.ModularFormsAmbient method), 22
EisensteinSubmodule (class in sage.modular.modform.eisenstein_submodule), 36
EisensteinSubmodule_eps (class in sage.modular.modform.eisenstein_submodule), 37
EisensteinSubmodule_g0_Q (class in sage.modular.modform.eisenstein_submodule), 38
EisensteinSubmodule_g1_Q (class in sage.modular.modform.eisenstein_submodule), 38
EisensteinSubmodule_gH_Q (class in sage.modular.modform.eisenstein_submodule), 38
EisensteinSubmodule_params (class in sage.modular.modform.eisenstein_submodule), 38
Ek_ZZ() (in module sage.modular.modform.eis_series_cython), 44
Element (sage.modular.modform.space.ModularFormsSpace attribute), 7
element() (sage.modular.modform.element.Newform method), 62
elliptic_curve() (sage.modular.modform.element.ModularFormElement_elliptic_curve method), 50
embedded_submodule() (sage.modular.modform.space.ModularFormsSpace method), 11

F
find_generators() (in module sage.modular.modform.find_generators), 80
find_in_space() (sage.modular.modform.space.ModularFormsSpace method), 11
free_module() (sage.modular.modform.ambient.ModularFormsAmbient method), 23

G
gen() (sage.modular.modform.space.ModularFormsSpace method), 12
gen_forms() (sage.modular.modform.find_generators.ModularFormsRing method), 76
generators() (sage.modular.modform.find_generators.ModularFormsRing method), 77
gens() (sage.modular.modform.space.ModularFormsSpace method), 12
group() (sage.modular.modform.element.ModularForm_abstract method), 51
group() (sage.modular.modform.find_generators.ModularFormsRing method), 79
group() (sage.modular.modform.space.ModularFormsSpace method), 13

H
half_integral_weight_modform_basis() (in module sage.modular.modform.half_integral), 73
has_character() (sage.modular.modform.space.ModularFormsSpace method), 13
has_cm() (sage.modular.modform.element.ModularForm_abstract method), 52
hecke_eigenvalue_field() (sage.modular.modform.element.Newform method), 63
hecke_module_of_level() (sage.modular.modform.ambient.ModularFormsAmbient method), 23
hecke_module_of_level() (sage.modular.modform.ambient_eps.ModularFormsAmbient_eps method), 28
hecke_operator_on_basis() (in module sage.modular.modform.hecke_operator_on_qexp), 66
hecke_operator_on_qexp() (in module sage.modular.modform.hecke_operator_on_qexp), 67
hecke_polynomial() (sage.modular.modform.ambient.ModularFormsAmbient method), 24
hecke_polynomial() (sage.modular.modform.cuspidal_submodule.CuspidalSubmodule_modsym_qexp
method), 35

I
integral_basis() (sage.modular.modform.space.ModularFormsSpace method), 13
is_ambient() (sage.modular.modform.ambient.ModularFormsAmbient method), 14
is_ambient() (sage.modular.modform.space.ModularFormsSpace method), 24
is_cuspidal() (sage.modular.modform.cuspidal_submodule.CuspidalSubmodule method), 33
is_cuspidal() (sage.modular.modform.element.Newform method), 63
is_cuspidal() (sage.modular.modform.space.ModularFormsSpace method), 14
is_eisenstein() (sage.modular.modform.space.ModularFormsSpace method), 14
is_ModularFormElement() (in module sage.modular.modform.element), 66
is_ModularFormsSpace() (in module sage.modular.modform.space), 20
J

j_invariant_qexp() *(in module sage.modular.modform.j_invariant)*, 81

L

L() *(sage.modular.modform.element.EisensteinSeries method)*, 46
level() *(sage.modular.modform.element.ModularForm_abstract method)*, 52
level() *(sage.modular.modform.numerical.NumericalEigenforms method)*, 69
level() *(sage.modular.modform.space.ModularFormsSpace method)*, 14
lseries() *(sage.modular.modform.element.ModularForm_abstract method)*, 52

M

M() *(sage.modular.modform.element.EisensteinSeries method)*, 46
modsymb_eigenspace() *(sage.modular.modform.element.Newform method)*, 63
modular_forms_of_weight() *(sage.modular.modform.find_generators.ModularFormsRing method)*, 79
modular_symbols() *(sage.modular.modform.ambient.ModularFormsAmbient method)*, 24
modular_symbols() *(sage.modular.modform.ambient_aps.ModularFormsAmbient_aps method)*, 24
modular_symbols() *(sage.modular.modform.ambient_R ambiguous ModularFormsAmbient_R method)*, 29
modular_symbols() *(sage.modular.modform.cuspidal_submodule.CuspidalSubmodule method)*, 34
modular_symbols() *(sage.modular.modform.eisenstein_submodule.EisensteinSubmodule method)*, 36
modular_symbols() *(sage.modular.modform.element.Newform method)*, 64
modular_symbols() *(sage.modular.modform.numerical.NumericalEigenforms method)*, 70
modular_symbols() *(sage.modular.modform.space.ModularFormsSpace method)*, 15
ModularForm_abstract *(class in sage.modular.modform.element)*, 50
ModularFormElement *(class in sage.modular.modform.element)*, 47
ModularFormElement_elliptic_curve *(class in sage.modular.modform.element)*, 49
ModularForms() *(in module sage.modular.modform.constructor)*, 1
ModularFormsClear_cache() *(in module sage.modular.modform.constructor)*, 4
ModularFormsAmbient *(class in sage.modular.modform.ambient)*, 21
ModularFormsAmbient_aps *(class in sage.modular.modform.ambient_aps)*, 27
ModularFormsAmbient_g0_Q *(class in sage.modular.modform.ambient_g0)*, 29
ModularFormsAmbient_g1_Q *(class in sage.modular.modform.ambient_g1)*, 30
ModularFormsAmbient_R *(class in sage.modular.modform.ambient_R)*, 31
ModularFormsRing *(class in sage.modular.modform.find_generators)*, 75
ModularFormsSpace *(class in sage.modular.modform.space)*, 7
ModularFormsSubmodule *(class in sage.modular.modform.submodule)*, 32
ModularFormsSubmoduleWithBasis *(class in sage.modular.modform.submodule)*, 32
module() *(sage.modular.modform.ambient.ModularFormsAmbient method)*, 24

N

new_eisenstein_series() *(sage.modular.modform.eisenstein_submodule.EisensteinSubmodule_params method)*, 40
new_level() *(sage.modular.modform.element.EisensteinSeries method)*, 46
new_submodule() *(sage.modular.modform.ambient.ModularFormsAmbient method)*, 25
new_submodule() *(sage.modular.modform.cuspidal_submodule.CuspidalSubmodule_modsym_qexp method)*, 36
new_submodule() *(sage.modular.modform.eisenstein_submodule.EisensteinSubmodule_modsym_qexp method)*, 40
new_submodule() *(sage.modular.modform.space.ModularFormsSpace method)*, 15
new_subspace() *(sage.modular.modform.space.ModularFormsSpace method)*, 15
Newform *(class in sage.modular.modform.element)*, 59
Newform() *(in module sage.modular.modform.constructor)*, 4
<table>
<thead>
<tr>
<th>Function</th>
<th>Module/Method</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>set_precision()</td>
<td>(sage.modular.modform.ambient.ModularFormsAmbient method)</td>
<td>26</td>
</tr>
<tr>
<td>set_precision()</td>
<td>(sage.modular.modform.space.ModularFormsSpace method)</td>
<td>17</td>
</tr>
<tr>
<td>span()</td>
<td>(sage.modular.modform.space.ModularFormsSpace method)</td>
<td>18</td>
</tr>
<tr>
<td>span_of_basis()</td>
<td>(sage.modular.modform.space.ModularFormsSpace method)</td>
<td>19</td>
</tr>
<tr>
<td>sturm_bound()</td>
<td>(sage.modular.modform.space.ModularFormsSpace method)</td>
<td>19</td>
</tr>
<tr>
<td>support()</td>
<td>(in module sage.modular.modform.numerical)</td>
<td>70</td>
</tr>
<tr>
<td>symsquare_lseries()</td>
<td>(sage.modular.modform.element.ModularForm_abstract method)</td>
<td>58</td>
</tr>
<tr>
<td>systems_of_abs()</td>
<td>(sage.modular.modform.numerical.NumericalEigenforms method)</td>
<td>70</td>
</tr>
<tr>
<td>systems_of_eigenvalues()</td>
<td>(sage.modular.modform.numerical.NumericalEigenforms method)</td>
<td>70</td>
</tr>
<tr>
<td>T</td>
<td></td>
<td></td>
</tr>
<tr>
<td>t()</td>
<td>(sage.modular.modform.element.EisensteinSeries method)</td>
<td>47</td>
</tr>
<tr>
<td>theta2_qexp()</td>
<td>(in module sage.modular.modform.theta)</td>
<td>81</td>
</tr>
<tr>
<td>theta_qexp()</td>
<td>(in module sage.modular.modform.theta)</td>
<td>82</td>
</tr>
<tr>
<td>twist()</td>
<td>(sage.modular.modform.element.ModularFormElement method)</td>
<td>48</td>
</tr>
<tr>
<td>twist()</td>
<td>(sage.modular.modform.element.Newform method)</td>
<td>64</td>
</tr>
<tr>
<td>V</td>
<td></td>
<td></td>
</tr>
<tr>
<td>valuation()</td>
<td>(sage.modular.modform.element.ModularForm_abstract method)</td>
<td>58</td>
</tr>
<tr>
<td>victor_miller_basis()</td>
<td>(in module sage.modular.modform.vm_basis)</td>
<td>72</td>
</tr>
<tr>
<td>W</td>
<td></td>
<td></td>
</tr>
<tr>
<td>weight()</td>
<td>(sage.modular.modform.element.ModularForm_abstract method)</td>
<td>58</td>
</tr>
<tr>
<td>weight()</td>
<td>(sage.modular.modform.numerical.NumericalEigenforms method)</td>
<td>70</td>
</tr>
<tr>
<td>weight()</td>
<td>(sage.modular.modform.space.ModularFormsSpace method)</td>
<td>20</td>
</tr>
</tbody>
</table>