Modular Forms

Release 9.7

The Sage Development Team

Sep 20, 2022

CONTENTS

1 Module List **1**
- 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 . . . 28
- 1.5 Modular Forms for $\Gamma_0(N)$ over $\mathbf{Q}$ . . . 31
- 1.6 Modular Forms for $\Gamma_1(N)$ and $\Gamma_H(N)$ over $\mathbf{Q}$ . . . 31
- 1.7 Modular Forms over a Non-minimal Base Ring . . . 33
- 1.8 Submodules of spaces of modular forms . . . 34
- 1.9 The Cuspidal Subspace . . . 34
- 1.10 The Eisenstein Subspace . . . 38
- 1.11 Eisenstein Series . . . 44
- 1.12 Eisenstein Series (optimized compiled functions) . . . 47
- 1.13 Elements of modular forms spaces . . . 47
- 1.14 Hecke Operators on q-expansions . . . 76
- 1.15 Numerical computation of newforms . . . 78
- 1.16 The Victor Miller Basis . . . 81
- 1.17 Compute spaces of half-integral weight modular forms . . . 84
- 1.18 Graded rings of modular forms . . . 85
- 1.19 q-expansion of j-invariant . . . 95
- 1.20 q-expansions of Theta Series . . . 95

2 Design Notes **97**
- 2.1 Design Notes . . . 97

3 Indices and Tables **99**

Python Module Index **101**

Index **103**

CHAPTER

ONE

MODULE LIST

1.1 Creating Spaces of Modular Forms

EXAMPLES:

```
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:

```
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:

```
sage: EisensteinForms(11,2)
Eisenstein subspace of dimension 1 of Modular Forms space of dimension 2 for␣
↪Congruence Subgroup Gamma0(11) of weight 2 over Rational Field
```

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

Create an ambient space of modular forms.

INPUT:

- `group` - A congruence subgroup or a Dirichlet character eps.
- `weight` - int, the weight, which must be an integer >= 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:

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

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

```
sage: ModularForms(1,12).dimension()
2
sage: ModularForms(11,4)
Modular Forms space of dimension 4 for Congruence Subgroup Gamma0(11) of weight 4␣
↪over Rational Field
```

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

```
sage: ModularForms(Gamma1(13),2)
Modular Forms space of dimension 13 for Congruence Subgroup Gamma1(13) of weight 2␣
↪over Rational Field
sage: ModularForms(Gamma1(13),2).dimension()
13
sage: [ModularForms(Gamma1(7),k).dimension() for k in [2,3,4,5]]
[5, 7, 9, 11]
sage: ModularForms(Gamma1(5),11).dimension()
12
```

We create a space with character:

```
sage: e = (DirichletGroup(13).0)^2
sage: e.order()
6
sage: M = ModularForms(e, 2); M
Modular Forms space of dimension 3, character [zeta6] and weight 2 over Cyclotomic␣
↪Field of order 6 and degree 2
sage: f = M.T(2).charpoly('x'); f
x^3 + (-2*zeta6 - 2)*x^2 - 2*zeta6*x + 14*zeta6 - 7
sage: f.factor()
(x - zeta6 - 2) * (x - 2*zeta6 - 1) * (x + zeta6 + 1)
```

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()
[
```

(continues on next page)

(continued from previous page)

```
]
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:

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

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

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

sage.modular.modform.constructor.**ModularForms_clear_cache**()

Clear the cache of modular forms.

EXAMPLES:

```
sage: M = ModularForms(37,2)
sage: sage.modular.modform.constructor._cache == {}
False
```

```
sage: sage.modular.modform.constructor.ModularForms_clear_cache()
sage: sage.modular.modform.constructor._cache
{}
```

sage.modular.modform.constructor.**Newform**(*identifier*, *group=None*, *weight=2*, *base_ring=Rational Field*, *names=None*)

INPUT:

- `identifier` - a canonical label, or the index of the specific newform desired
- `group` - the congruence subgroup of the newform
- `weight` - the weight of the newform (default 2)
- `base_ring` - the base ring
- `names` - if the newform has coefficients in a number field, a generator name must be specified

EXAMPLES:

```
sage: Newform('67a', names='a')
q + 2*q^2 - 2*q^3 + 2*q^4 + 2*q^5 + O(q^6)
```

(continues on next page)

(continued from previous page)

```
sage: Newform('67b', names='a')
q + a1*q^2 + (-a1 - 3)*q^3 + (-3*a1 - 3)*q^4 - 3*q^5 + O(q^6)
```

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

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

INPUT:

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

EXAMPLES:

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

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

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

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

Given a group, level, weight, and base_ring as input by the user, return a canonicalized version of them, where level is a Sage integer, group really is a group, weight is a Sage integer, and base_ring a Sage ring. Note that we can't just get the level from the group, because we have the convention that the character for Gamma1(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:

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

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

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

EXAMPLES:

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

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

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

1.2 Generic spaces of modular forms

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

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

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

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

```
sage: ModularForms(11,2,base_ring=GF(13)).base_ring()
Finite Field of size 13
```

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

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

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

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

An example in characteristic 7:

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

AUTHORS:

- William Stein (2007): first version

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

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:

```
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:

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

Spaces of forms with nontrivial character:

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

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

(continues on next page)

(continued from previous page)

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

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

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

If the level is one then the character is trivial.

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

cuspidal_submodule()

Return the cuspidal submodule of self.

EXAMPLES:

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

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

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

We check that a bug noticed on 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),
```

(continues on next page)

(continued from previous page)

```
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()
[
1 + 12*q^2 + 12*q^3 + 12*q^4 + 12*q^5 + O(q^6),
q - 2*q^2 - q^3 + 2*q^4 + q^5 + O(q^6)
]
```

```
sage: M = ModularForms(Gamma1(6),4)
sage: M.basis()
[
q - 2*q^2 - 3*q^3 + 4*q^4 + 6*q^5 + O(q^6),
1 + O(q^6),
q - 8*q^4 + 126*q^5 + O(q^6),
q^2 + 9*q^4 + O(q^6),
q^3 + O(q^6)
]
sage: M.echelon_form().basis()
[
1 + O(q^6),
q + 94*q^5 + O(q^6),
q^2 + 36*q^5 + O(q^6),
q^3 + O(q^6),
q^4 - 4*q^5 + O(q^6)
]
```

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

```
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)
]
sage: F = M.span_of_basis([M.0 + 1/3*M.1, M.2 + M.3]); F.basis()
```

(continues on next page)

(continued from previous page)

```
[
q + 1/3*q^2 + 5/3*q^3 - 16/3*q^4 - 13/3*q^5 + O(q^6),
1 + q + 9*q^2 + 28*q^3 + 73*q^4 + 126*q^5 + O(q^6)
]
sage: E = F.echelon_form(); E.basis()
[
1 + 26/3*q^2 + 79/3*q^3 + 235/3*q^4 + 391/3*q^5 + O(q^6),
q + 1/3*q^2 + 5/3*q^3 - 16/3*q^4 - 13/3*q^5 + O(q^6)
]
```

eisenstein_series()

Compute the Eisenstein series associated to this space.

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

EXAMPLES:

```
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.

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)
]
```

group()

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

EXAMPLES:

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

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

Note that $\Gamma_1(1)$ and $\Gamma_0(1)$ are replaced by $\mathrm{SL}_2(\mathbf{Z})$.

```
sage: CuspForms(Gamma1(1),12).group()
Modular Group SL(2,Z)
sage: CuspForms(SL2Z,12).group()
Modular Group SL(2,Z)
```

has_character()

Return True if this space of modular forms has a specific character.

This is True exactly when the character() function does not return None.

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

```
sage: CuspForms(Gamma0(11),2).has_character()
True
```

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

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

integral_basis()

Return an integral basis for this space of modular forms.

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

```
sage: m = ModularForms(97,2,prec=10)
sage: s = m.cuspidal_subspace()
sage: s.integral_basis()
[
q + 2*q^7 + 4*q^8 - 2*q^9 + O(q^10),
q^2 + q^4 + q^7 + 3*q^8 - 3*q^9 + 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 + 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)
]
```

Here's another example where there is a big gap in the valuations:

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

is_ambient()

Return True if this an ambient space of modular forms.

EXAMPLES:

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

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

(continues on next page)

(continued from previous page)

```
False
```

is_cuspidal()

Return True if this space is cuspidal.

EXAMPLES:

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

is_eisenstein()

Return True if this space is Eisenstein.

EXAMPLES:

```
sage: M = ModularForms(Gamma0(11), 2).new_submodule()
sage: M.is_eisenstein()
False
sage: M.eisenstein_submodule().is_eisenstein()
True
```

level()

Return the level of self.

EXAMPLES:

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

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

new_subspace(*p=None*)

Synonym for new_submodule.

EXAMPLES:

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

newforms(*names=None*)

Return all newforms in the cuspidal subspace of self.

EXAMPLES:

```
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()
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:

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

(continues on next page)

(continued from previous page)

```
[
q - 24*q^2 + 252*q^3 - 1472*q^4 + 4830*q^5 - 6048*q^6 - 16744*q^7 + O(q^8)
]
```

q_echelon_basis(*prec=None*)

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

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

EXAMPLES:

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

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

q_expansion_basis(*prec=None*)

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

INPUT:

- `prec` - integer (>=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:

```
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),
```

(continues on next page)

(continued from previous page)

```
q^2 - 48*q^3 + 1080*q^4 - 15040*q^5 + 143820*q^6 - 985824*q^7 + O(q^8)
]
```

An example which used to be buggy:

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

q_integral_basis(*prec=None*)

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

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

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

EXAMPLES:

```
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()
[
```

(continues on next page)

(continued from previous page)

```
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
```

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

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

span_of_basis(*B*)

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

EXAMPLES:

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

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

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

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

sturm_bound(*M=None*)

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

EXAMPLES:

```
sage: S37=CuspForms(37,2)
sage: S37.sturm_bound()
8
sage: M = ModularForms(11,2)
sage: M.sturm_bound()
3
sage: ModularForms(Gamma1(15),2).sturm_bound()
33

sage: CuspForms(Gamma1(144), 3).sturm_bound()
3457
sage: CuspForms(DirichletGroup(144).1^2, 3).sturm_bound()
73
sage: CuspForms(Gamma0(144), 3).sturm_bound()
73
```

REFERENCES:

- [Stu1987]

NOTE:

Kevin Buzzard pointed out to me (William Stein) in Fall 2002 that the above bound is fine for Gamma1 with character, as one sees by taking a power of f. More precisely, if $f \cong 0 \pmod{p}$ for first s coefficients, then $f^r = 0 \pmod{p}$ for first sr coefficients. Since the weight of f^r is $r\text{weight}(f)$, it follows that if $s \geq$ the Sturm bound for Γ_0 at weight(f), then f^r has valuation large enough to be forced to be 0 at $r\cdot$ weight(f) by Sturm bound (which is valid if we choose r right). Thus $f \cong 0 \pmod{p}$. Conclusion: For Γ_1 with fixed character, the Sturm bound is *exactly* the same as for Γ_0. A key point is that we are finding $\mathbf{Z}[\varepsilon]$ generators for the Hecke algebra here, not $\mathbf{Z}$-generators. So if one wants generators for the Hecke algebra over $\mathbf{Z}$, this bound is wrong.

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

weight()
Return the weight of this space of modular forms.

EXAMPLES:

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

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

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

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:

```
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:

```
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 χ is quadratic.

```
sage: chi = DirichletGroup(25,QQ).0; chi
Dirichlet character modulo 25 of conductor 5 mapping 2 |--> -1
sage: n = ModularForms(chi,2); n
```

(continues on next page)

(continued from previous page)

```
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:

```
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:

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

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

Bases: *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:

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

change_ring(*base_ring*)

Change the base ring of this space of modular forms.

INPUT:

- R - ring

EXAMPLES:

```
sage: M = ModularForms(Gamma0(37),2)
sage: M.basis()
```

(continues on next page)

(continued from previous page)

```
[
q + q^3 - 2*q^4 + O(q^6),
q^2 + 2*q^3 - 2*q^4 + q^5 + O(q^6),
1 + 2/3*q + 2*q^2 + 8/3*q^3 + 14/3*q^4 + 4*q^5 + O(q^6)
]
```

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

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

cuspidal_submodule()

Return the cuspidal submodule of this ambient module.

EXAMPLES:

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

dimension()

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

EXAMPLES:

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

eisenstein_params()

Return parameters that define all Eisenstein series in self.

OUTPUT: an immutable Sequence

EXAMPLES:

```
sage: m = ModularForms(Gamma0(22), 2)
sage: v = m.eisenstein_params(); v
[(Dirichlet character modulo 22 of conductor 1 mapping 13 |--> 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 + q^3 + 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(8) of weight␣
↪3 over Rational Field
sage: ModularForms(Gamma1(4), 3).hecke_module_of_level(9)
Traceback (most recent call last):
...
ValueError: N (=9) must be a divisor or a multiple of the level of self (=4)
```

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

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

EXAMPLES:

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

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

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

is_ambient()

Return True if this an ambient space of modular forms.

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

EXAMPLES:

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

modular_symbols(*sign=0*)

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

EXAMPLES:

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

prec(*new_prec=None*)

Set or get default initial precision for printing modular forms.

INPUT:

- `new_prec` - positive integer (default: None)

OUTPUT: if new_prec is None, returns the current precision.

EXAMPLES:

```
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:

```
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:

```
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()
```

(continues on next page)

(continued from previous page)

```
[
1 + 60*q^3 - 120*q^4 + O(q^5),
q + 6*q^3 - 9*q^4 + O(q^5),
q^2 - 4*q^3 + 12*q^4 + O(q^5)
]
```

1.4 Modular Forms with Character

EXAMPLES:

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

sage: S = M.cuspidal_submodule(); S
Cuspidal subspace of dimension 1 of Modular Forms space of dimension 3, character␣
↪[zeta6] and weight 2 over Cyclotomic Field of order 6 and degree 2
sage: S.modular_symbols()
Modular Symbols subspace of dimension 2 of Modular Symbols space of dimension 4 and␣
↪level 13, weight 2, character [zeta6], sign 0, over Cyclotomic Field of order 6 and␣
↪degree 2
```

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

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

Notice that the base ring is “minimized”:

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

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

```
sage: e = DirichletGroup(225).1
sage: e.order()
20
sage: e.base_ring()
Cyclotomic Field of order 60 and degree 16
```

(continues on next page)

(continued from previous page)

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

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

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

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

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

eisenstein_submodule()

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

EXAMPLES:

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

hecke_module_of_level(*N*)

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

EXAMPLES:

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

modular_symbols(*sign=0*)

Return corresponding space of modular symbols with given sign.

EXAMPLES:

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

1.5 Modular Forms for $\Gamma_0(N)$ over $\mathbf{Q}$

class sage.modular.modform.ambient_g0.**ModularFormsAmbient_g0_Q**(*level*, *weight*)

Bases: *sage.modular.modform.ambient.ModularFormsAmbient*

A space of modular forms for $\Gamma_0(N)$ over $\mathbf{Q}$.

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 $\mathbf{Q}$

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),
```

(continues on next page)

(continued from previous page)

```
1 + 12/5*q + 36/5*q^2 + 48/5*q^3 + 84/5*q^4 + 72/5*q^5 + 144/5*q^6 + 96/5*q^7 + O(q^8)
]
```

class sage.modular.modform.ambient_g1.**ModularFormsAmbient_g1_Q**(*level*, *weight*, *eis_only*)

Bases: *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
```

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

eisenstein_submodule()

Return the Eisenstein submodule of this modular forms space.

EXAMPLES:

```
sage: E = ModularForms(GammaH(100, [29]),3).eisenstein_submodule(); E
Eisenstein subspace of dimension 24 of Modular Forms space of dimension 72 for␣
↪Congruence Subgroup Gamma_H(100) with H generated by [29] of weight 3 over␣
↪Rational Field
```

(continues on next page)

(continued from previous page)

```
sage: type(E)
<class 'sage.modular.modform.eisenstein_submodule.EisensteinSubmodule_gH_Q_with_
↪category'>
```

1.7 Modular Forms over a Non-minimal Base Ring

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

Bases: *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
```

change_ring(*R*)

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

EXAMPLES:

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

cuspidal_submodule()

Return the cuspidal subspace of this space.

EXAMPLES:

```
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:

```
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:

```
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
```

(continues on next page)

(continued from previous page)

```
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*)

Bases: *sage.modular.modform.submodule.ModularFormsSubmodule*

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

change_ring(*R*)

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

EXAMPLES:

```
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:

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

modular_symbols(*sign=0*)

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

EXAMPLES:

```
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
```

(continues on next page)

(continued from previous page)

```
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)
```

(continues on next page)

(continued from previous page)

```
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 $\mathbf{Q}$.

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 $\mathbf{Q}$.

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 $\mathbf{Q}$.

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 $\mathbf{Q}$.

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:

```
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
sage: M.sign()
0

sage: M = E.modular_symbols(sign=-1); M
Modular Symbols subspace of dimension 0 of Modular Symbols space of
dimension 1 for Gamma_0(11) of weight 2 with sign -1 over Rational Field

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

(continues on next page)

(continued from previous page)

```
sage: eps = DirichletGroup(13).0
sage: E = EisensteinForms(eps^2, 2)
sage: E.modular_symbols()
Modular Symbols subspace of dimension 2 of Modular Symbols space of dimension 4␣
↪and level 13, weight 2, character [zeta6], sign 0, over Cyclotomic Field of␣
↪order 6 and degree 2

sage: E = EisensteinForms(eps, 1); E
Eisenstein subspace of dimension 1 of Modular Forms space of character [zeta12]␣
↪and weight 1 over Cyclotomic Field of order 12 and degree 4
sage: E.modular_symbols()
Traceback (most recent call last):
...
ValueError: the weight must be at least 2
```

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:

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

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)
]
sage: M.eisenstein_subspace().T(2).matrix().fcp()
(x + 2*zeta3 + 1) * (x + zeta3 + 2) * (x - zeta3 - 2)^2 * (x - 2*zeta3 - 1)^2
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

sage: M.basis()
[
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
↪Congruence Subgroup Gamma0(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
↪Congruence Subgroup Gamma0(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)
```

(continues on next page)

(continued from previous page)

```
]
sage: EisensteinForms(1,24).eisenstein_series()
[
236364091/131040 + q + 8388609*q^2 + 94143178828*q^3 + 70368752566273*q^4 +␣
↪11920928955078126*q^5 + O(q^6)
]
sage: EisensteinForms(5,4).eisenstein_series()
[
1/240 + q + 9*q^2 + 28*q^3 + 73*q^4 + 126*q^5 + O(q^6),
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 + 2)*q^2 + (zeta6 + 2)*q^3 + O(q^4),
-1/7*zeta6 - 2/7 + q + (2*zeta6 - 1)*q^2 + (-3*zeta6 + 1)*q^3 + O(q^4),
q + (zeta6 + 1)*q^2 + (-zeta6 + 3)*q^3 + O(q^4)
]

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

sage: M = ModularForms(19,3).eisenstein_subspace()
sage: M.eisenstein_series()
[
]

sage: M = ModularForms(DirichletGroup(13).0, 1)
sage: M.eisenstein_series()
[
-1/13*zeta12^3 + 6/13*zeta12^2 + 4/13*zeta12 + 2/13 + q + (zeta12 + 1)*q^2 +␣
↪zeta12^2*q^3 + (zeta12^2 + zeta12 + 1)*q^4 + (-zeta12^3 + 1)*q^5 + O(q^6)
]

sage: M = ModularForms(GammaH(15, [4]), 4)
sage: M.eisenstein_series()
[
1/240 + q + 9*q^2 + 28*q^3 + 73*q^4 + 126*q^5 + O(q^6),
1/240 + q^3 + O(q^6),
```

(continues on next page)

(continued from previous page)

```
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)
]
```

new_eisenstein_series()

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

EXAMPLES:

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

new_submodule(*p=None*)

Return the new submodule of self.

EXAMPLES:

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

parameters()

Return a list of parameters for each Eisenstein series spanning self. That is, for each such series, return a triple of the form (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 7 |--> 1, 5 |--> 1,␣
↪Dirichlet character modulo 12 of conductor 1 mapping 7 |--> 1, 5 |--> 1, 12)

sage: pars = ModularForms(DirichletGroup(24).0,3).eisenstein_submodule().
↪parameters()
```

(continues on next page)

(continued from previous page)

```
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, 1), (-1, 1, 1), 2),
((1, 1, 1), (-1, 1, 1), 3),
((1, 1, 1), (-1, 1, 1), 6),
((-1, 1, 1), (1, 1, 1), 1),
((-1, 1, 1), (1, 1, 1), 2),
((-1, 1, 1), (1, 1, 1), 3),
((-1, 1, 1), (1, 1, 1), 6)]
sage: EisensteinForms(DirichletGroup(24).0,1).parameters()
[(Dirichlet character modulo 24 of conductor 1 mapping 7 |--> 1, 13 |--> 1, 17␣
↪|--> 1, Dirichlet character modulo 24 of conductor 4 mapping 7 |--> -1, 13 |--
↪> 1, 17 |--> 1, 1), (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)]
```

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

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

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 ζ_m generates M, not L.)

EXAMPLES:

```
sage: L = CyclotomicField(12) ; N = CyclotomicField(33) ; M = CyclotomicField(132)
sage: z, n = sage.modular.modform.eisenstein_submodule.cyclotomic_restriction(L,N)
sage: n
2

sage: z(L.0)
-zeta33^19*x
sage: z(L.0)(M.0)
zeta132^11

sage: z(L.0^3-L.0+1)
(zeta33^19 + zeta33^8)*x + 1
sage: z(L.0^3-L.0+1)(M.0)
zeta132^33 - zeta132^11 + 1
sage: z(L.0^3-L.0+1)(M.0) - M(L.0^3-L.0+1)
0
```

sage.modular.modform.eisenstein_submodule.**cyclotomic_restriction_tower**(L, K)

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

INPUT:

an element alpha in L

OUTPUT:

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

EXAMPLES:

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

1.11 Eisenstein Series

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

Compute and return a list of all parameters (χ, ψ, 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 $\chi(-1) * \psi(-1) = (-1)^k$.

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

EXAMPLES:

```
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),
```

(continues on next page)

(continued from previous page)

```
((1, 1), (1, zeta4), 1),
((1, 1), (1, zeta4), 3),
((1, 1), (-1, -1), 1),
((1, 1), (1, -zeta4), 1),
((1, 1), (1, -zeta4), 3),
((-1, 1), (1, -1), 1)]

sage: sage.modular.modform.eis_series.compute_eisenstein_params(DirichletGroup(15).
↪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_asymp_coeffs=40*)

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

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

INPUT:

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

OUTPUT:

The L-series of the Eisenstein series.

EXAMPLES:

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

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

Now with higher precision:

```
sage: L = eisenstein_series_lseries(20, prec=200)
sage: L(2)
-5.0235535164599797471968418348135050804419155747868718371029
```

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 $\mathrm{SL}_2(\mathbf{Z})$ to precision prec in the ring K. Three normalizations are available, depending on the parameter `normalization`; the default normalization is the one for which the linear coefficient is 1.

INPUT:

- `k` - an even positive integer
- `prec` - (default: 10) a nonnegative integer
- `K` - (default: $\mathbf{Q}$) 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 σ 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)

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

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

INPUT:

- k – int
- prec – 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 prec of the weight k Eisenstein series, as a FLINT Fmpz_poly object, normalised so the coefficients are integers with no common factor.

Used internally by the functions *eisenstein_series_qexp()* and *victor_miller_basis()*; see the docstring of the former for further details.

EXAMPLES:

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

1.13 Elements of modular forms spaces

Class hierarchy:

- *ModularForm_abstract*
  - *Newform*
    - *ModularFormElement_elliptic_curve*
  - *ModularFormElement*
    - *EisensteinSeries*
- *GradedModularFormElement*

AUTHORS:

- William Stein (2004-2008): first version
- David Ayotte (2021-06): GradedModularFormElement class

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()
[
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
[
```

(continues on next page)

(continued from previous page)

```
-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, 2], [60, 3], [60,␣
↪2], [60, 2], [60, 2]]
```

parameters()

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

EXAMPLES:

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

psi()

Return the parameter psi associated to self.

EXAMPLES:

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

t()

Return the parameter t associated to self.

EXAMPLES:

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

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

Bases: `sage.structure.element.ModuleElement`

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

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

EXAMPLES:

```
sage: M = ModularFormsRing(1)
sage: D = CuspForms(1, 12).0
sage: M(D).parent()
Ring of Modular Forms for Modular Group SL(2,Z) over Rational Field
```

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

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

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

```
sage: (E4 + D).parent()
Ring of Modular Forms for Modular Group SL(2,Z) over Rational Field
sage: M([E4, D]) == E4 + D
True
```

Graded modular forms elements for congruence subgroups are also supported:

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

derivative(*name='E2'*)

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

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

INPUT:

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

OUTPUT: a `sage.modular.quasimodform.element.QuasiModularFormsElement`

EXAMPLES:

```
sage: M = ModularFormsRing(1)
sage: E4 = M.0; E6 = M.1
sage: dE4 = E4.derivative(); dE4
240*q + 4320*q^2 + 20160*q^3 + 70080*q^4 + 151200*q^5 + O(q^6)
sage: dE4.parent()
Ring of Quasimodular Forms for Modular Group SL(2,Z) over Rational Field
sage: dE4.is_modular_form()
False
```

group()

Return the group for which `self` is a modular form.

EXAMPLES:

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

homogeneous_component(*weight*)

Return the homogeneous component of the given graded modular form.

INPUT:

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

EXAMPLES:

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

is_homogeneous()

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

An alias of this method is `is_modular_form`

EXAMPLES:

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

is_modular_form()

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

An alias of this method is `is_modular_form`

EXAMPLES:

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

is_one()

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

EXAMPLES:

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

is_zero()

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

EXAMPLES:

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

q_expansion(*prec=None*)

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

An alias of this method is `qexp`.

EXAMPLES:

```
sage: M = ModularFormsRing(1)
sage: zer = M(0); zer.q_expansion()
0
sage: M(5/7).q_expansion()
5/7
sage: E4 = M.0; E4
1 + 240*q + 2160*q^2 + 6720*q^3 + 17520*q^4 + 30240*q^5 + O(q^6)
sage: E6 = M.1; E6
1 - 504*q - 16632*q^2 - 122976*q^3 - 532728*q^4 - 1575504*q^5 + O(q^6)
sage: F = E4 + E6; F
2 - 264*q - 14472*q^2 - 116256*q^3 - 515208*q^4 - 1545264*q^5 + O(q^6)
sage: F.q_expansion()
2 - 264*q - 14472*q^2 - 116256*q^3 - 515208*q^4 - 1545264*q^5 + O(q^6)
sage: F.q_expansion(10)
2 - 264*q - 14472*q^2 - 116256*q^3 - 515208*q^4 - 1545264*q^5 - 3997728*q^6 -
↪8388672*q^7 - 16907400*q^8 - 29701992*q^9 + O(q^10)
```

qexp(*prec=None*)

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

An alias of this method is `qexp`.

EXAMPLES:

```
sage: M = ModularFormsRing(1)
sage: zer = M(0); zer.q_expansion()
0
sage: M(5/7).q_expansion()
5/7
sage: E4 = M.0; E4
1 + 240*q + 2160*q^2 + 6720*q^3 + 17520*q^4 + 30240*q^5 + O(q^6)
sage: E6 = M.1; E6
1 - 504*q - 16632*q^2 - 122976*q^3 - 532728*q^4 - 1575504*q^5 + O(q^6)
sage: F = E4 + E6; F
2 - 264*q - 14472*q^2 - 116256*q^3 - 515208*q^4 - 1545264*q^5 + O(q^6)
sage: F.q_expansion()
2 - 264*q - 14472*q^2 - 116256*q^3 - 515208*q^4 - 1545264*q^5 + O(q^6)
sage: F.q_expansion(10)
2 - 264*q - 14472*q^2 - 116256*q^3 - 515208*q^4 - 1545264*q^5 - 3997728*q^6 -
↪8388672*q^7 - 16907400*q^8 - 29701992*q^9 + O(q^10)
```

serre_derivative()

Return the Serre derivative of the given graded modular form.

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

EXAMPLES:

```
sage: M = ModularFormsRing(1)
sage: E4 = M.0
sage: E6 = M.1
sage: DE4 = E4.serre_derivative(); DE4
-1/3 + 168*q + 5544*q^2 + 40992*q^3 + 177576*q^4 + 525168*q^5 + O(q^6)
sage: DE4 == (-1/3) * E6
```

(continues on next page)

(continued from previous page)

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

to_polynomial(*names='x'*, *gens=None*)

Return a polynomial $P(x_0, ..., x_n)$ such that $P(g_0, ..., g_n)$ is equal to `self` where $g_0, ..., g_n$ is a list of generators of the parent.

INPUT:

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

OUTPUT: A polynomial in the variables `names`

EXAMPLES:

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

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

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

weight()

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

EXAMPLES:

```
sage: D = ModularForms(1,12).0; M = ModularFormsRing(1)
sage: M(D).weight()
12
sage: M.zero().weight()
0
sage: e4 = ModularForms(1,4).0
sage: (M(D)+e4).weight()
```

(continues on next page)

(continued from previous page)

```
Traceback (most recent call last):
...
ValueError: the given graded form is not homogeneous (not a modular form)
```

weights_list()

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

EXAMPLES:

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

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

Bases: *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 ϵ and q-expansion

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

then the twist by χ is a modular form f_χ 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_χ as an element of the space of modular forms for $\Gamma_1(N)$ with character $\epsilon\chi^2$.

EXAMPLES:

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

Modular forms without character are supported:

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

```
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**(*parent*, *E*)

Bases: *sage.modular.modform.element.Newform*

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

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:

```
sage: EllipticCurve('57a1').newform().atkin_lehner_eigenvalue()
1
sage: EllipticCurve('57b1').newform().atkin_lehner_eigenvalue()
-1
```

(continues on next page)

(continued from previous page)

```
sage: EllipticCurve('57b1').newform().atkin_lehner_eigenvalue(19)
1
```

elliptic_curve()

Return elliptic curve associated to self.

EXAMPLES:

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

class sage.modular.modform.element.**ModularForm_abstract**

Bases: `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)
```

character(*compute=True*)

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

EXAMPLES:

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

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

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:

```
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
```

coefficient(n)

Return the n-th coefficient of the q-expansion of self.

INPUT:

- `n` (int, Integer) - A non-negative integer.

EXAMPLES:

```
sage: f = ModularForms(1, 12).0; f
q - 24*q^2 + 252*q^3 - 1472*q^4 + 4830*q^5 + O(q^6)
sage: f.coefficient(0)
0
sage: f.coefficient(1)
1
sage: f.coefficient(2)
-24
sage: f.coefficient(3)
252
sage: f.coefficient(4)
-1472
```

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:

```
sage: ModularForms(Gamma1(11), 2).gen(0).group()
Congruence Subgroup Gamma1(11)
```

has_cm()

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

OUTPUT:

Boolean

See also:

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

EXAMPLES:

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

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

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

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

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

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

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

is_homogeneous()

Return true. For compatibility with elements of a graded modular forms ring.

An alias of this method is `is_modular_form`.

See also:

meth $sage.modular.modform.element.GradedModularFormElement.is_homogeneous$

EXAMPLES:

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

is_modular_form()

Return true. For compatibility with elements of a graded modular forms ring.

An alias of this method is `is_modular_form`.

See also:

meth $sage.modular.modform.element.GradedModularFormElement.is_homogeneous$

EXAMPLES:

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

level()

Return the level of self.

EXAMPLES:

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

lseries(*embedding=0*, *prec=53*, *max_imaginary_part=0*, *max_asymp_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 $\mathbf{C}$, 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)
- `prec` - integer (bits precision). Default: 53.
- `max_imaginary_part` - real number. Default: 0.
- `max_asymp_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:

```
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)
```

(continues on next page)

(continued from previous page)

```
sage: L(1)
0.0884317737041015
sage: L(0.5)
0.0296568512531983
```

As a consistency check, we verify that the functional equation holds:

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

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

```
sage: f = Newforms(43, names='a')[1]
sage: K = f.hecke_eigenvalue_field()
sage: phi1, phi2 = K.embeddings(CC)
sage: L = f.lseries(embedding=phi1)
sage: L
L-series associated to the cusp form q + a1*q^2 - a1*q^3 + (-a1 + 2)*q^5 + O(q^
↪6), a1=-1.41421356237310
sage: L(1)
0.620539857407845
sage: L = f.lseries(embedding=1)
sage: L(1)
0.921328017272472
```

An example with a non-real coefficient field ($\mathbf{Q}(\zeta_3)$ in this case):

```
sage: f = Newforms(Gamma1(13), 2, names='a')[0]
sage: f.lseries(embedding=0)(1)
0.298115272465799 - 0.0402203326076734*I
sage: f.lseries(embedding=1)(1)
0.298115272465799 + 0.0402203326076732*I
```

We compute with the L-series of the Eisenstein series E_4:

```
sage: f = ModularForms(1,4).0
sage: L = f.lseries()
sage: L(1)
-0.0304484570583933
sage: L = eisenstein_series_lseries(4)
sage: L(1)
-0.0304484570583933
```

Consistency check with delta_lseries (which computes coefficients in pari):

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

We check that trac ticket #5262 is fixed:

```
sage: E = EllipticCurve('37b2')
sage: h = Newforms(37)[1]
sage: Lh = h.lseries()
sage: LE = E.lseries()
sage: Lh(1), LE(1)
(0.725681061936153, 0.725681061936153)
sage: CuspForms(1, 30).0.lseries().eps
-1.00000000000000
```

We check that trac ticket #25369 is fixed:

```
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):

```
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)
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 $Gamma_0(N)$
- `M` – an element of $\Gamma_0(N)$

- prec – (default: 53) the working precision in bits. If f is a normalised eigenform, then the output is correct to approximately this number of bits.

OUTPUT:

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

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

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)
-0.634604652139776 - 1.45881661693850*I

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

If E is an elliptic curve over $\mathbf{Q}$ 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
sage: omega1, omega2 = E.period_lattice().basis()
sage: -2/5*omega1 + omega2
0.634604652139777 + 1.45881661693850*I
```

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

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

```
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)*omega1 + t(3/11)*2*omega2.imag()*I
0.634604652139777 + 1.45881661693850*I
```

ALGORITHM:

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

Since the actual precision is finite, the output is currently *not* guaranteed to be correct to `prec` bits of precision.

petersson_norm(*embedding=0*, *prec=53*)

Compute the Petersson scalar product of f with itself:

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

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

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

INPUT:

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

EXAMPLES:

```
sage: CuspForms(1, 16).0.petersson_norm()
verbose -1 (...: dokchitser.py, __call__) Warning: Loss of 2 decimal digits due␣
↪to cancellation
2.16906134759063e-6
```

The Petersson norm depends on a choice of embedding:

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

prec()

Return the precision to which self.q_expansion() is currently known. Note that this may be 0.

EXAMPLES:

```
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 $O(q^{\mathrm{prec}})$. This function takes one argument, which is the integer prec.

EXAMPLES:

We compute the cusp form Δ:

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

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

```
sage: f = CuspForms(23,2).0
sage: f.q_expansion()
q - q^3 - q^4 + O(q^6)
sage: f.q_expansion(10)
q - q^3 - q^4 - 2*q^6 + 2*q^7 - q^8 + 2*q^9 + O(q^10)
sage: f.q_expansion(2)
q + O(q^2)
sage: f.q_expansion(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()

EXAMPLES:

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

serre_derivative()

Return the Serre derivative of the given modular form.

If `self` is of weight k, then the returned modular form will be of weight $k + 2$.

EXAMPLES:

```
sage: E4 = ModularForms(1, 4).0
sage: E6 = ModularForms(1, 6).0
sage: DE4 = E4.serre_derivative(); DE4
-1/3 + 168*q + 5544*q^2 + 40992*q^3 + 177576*q^4 + 525168*q^5 + O(q^6)
sage: DE6 = E6.serre_derivative(); DE6
-1/2 - 240*q - 30960*q^2 - 525120*q^3 - 3963120*q^4 - 18750240*q^5 + O(q^6)
sage: Del = ModularForms(1, 12).0 # Modular discriminant
sage: Del.serre_derivative()
0
sage: f = ModularForms(DirichletGroup(5).0, 1).0
sage: Df = f.serre_derivative(); Df
-1/12 + (-11/12*zeta4 + 19/4)*q + (11/6*zeta4 + 59/3)*q^2 + (-41/3*zeta4 + 239/
→6)*q^3 + (31/4*zeta4 + 839/12)*q^4 + (-251/12*zeta4 + 459/4)*q^5 + O(q^6)
```

(continues on next page)

(continued from previous page)

```
```

The Serre derivative raises the weight of a modular form by 2:

```
sage: DE4.weight()
6
sage: DE6.weight()
8
sage: Df.weight()
3
```

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

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

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

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

INPUT:

- chi – Dirichlet character to twist by, or None (default None, interpreted as the trivial character).
- embedding – embedding of the coefficient field into $\mathbf{R}$ or $\mathbf{C}$, 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:

```
sage: CuspForms(1, 12).0.symsquare_lseries()(22)
0.999645711124771
```

An example twisted by a nontrivial character:

```
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 $\mathbf{Q}$:

```
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:

```
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(Gamma1(9),2).6).weight()
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()
```

(continues on next page)

(continued from previous page)

```
Newform abelian subvariety 14a of dimension 1 of J0(14)
sage: Newforms(1, 12)[0].abelian_variety()
Traceback (most recent call last):
...
TypeError: f must have weight 2
```

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

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

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

See *atkin_lehner_eigenvalue()* for further details.

EXAMPLES:

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

The above computation can also be done exactly:

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

We can compute the eigenvalue of W_{p^e} 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 λ_d, using the conventions of [AL1978] for the weight k action, which imply that λ_d has complex absolute value 1. However, with these conventions λ_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 ε 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:

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

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)
sage: f.atkin_lehner_eigenvalue(5)
Traceback (most recent call last):
...
ValueError: Unable to compute square root. Try specifying an embedding into a␣
↪larger ring
sage: L = f.hecke_eigenvalue_field(); x = polygen(QQ); M.<sqrt5> = L.
↪extension(x^2 - 5)
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 random, of course.)

EXAMPLES:

```
sage: ls = Newforms(38,4,names='a')
sage: ls[0]
q - 2*q^2 - 2*q^3 + 4*q^4 - 9*q^5 + O(q^6)
sage: ls # random
[q - 2*q^2 - 2*q^3 + 4*q^4 - 9*q^5 + O(q^6),
q - 2*q^2 + (-a1 - 2)*q^3 + 4*q^4 + (2*a1 + 10)*q^5 + O(q^6),
q + 2*q^2 + (1/2*a2 - 1)*q^3 + 4*q^4 + (-3/2*a2 + 12)*q^5 + O(q^6)]
sage: type(ls[0])
<class 'sage.modular.modform.element.Newform'>
sage: ls[2][3].minpoly()
x^2 - 9*x + 2
sage: ls2 = [ x.element() for x in ls ]
sage: ls2 # 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(ls2[0])
<class 'sage.modular.modform.cuspidal_submodule.CuspidalSubmodule_g0_Q_with_
↪category.element_class'>
sage: ls2[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:

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

is_cuspidal()

Return True. For compatibility with elements of modular forms spaces.

EXAMPLES:

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

local_component(*p*, *twist_factor=None*)

Calculate the local component at the prime p of the automorphic representation attached to this newform. For more information, see the documentation of the `LocalComponent()` function.

EXAMPLES:

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

minimal_twist(*p=None*)

Compute a pair (g, chi) such that $g = f \otimes \chi$, where f is this newform and χ is a Dirichlet character, such that g has level as small as possible. If the optional argument p is given, consider only twists by Dirichlet characters of p-power conductor.

EXAMPLES:

```
sage: f = Newforms(575, 2, names='a')[4]
sage: g, chi = f.minimal_twist(5)
sage: g
q + a*q^2 - a*q^3 - 2*q^4 + (1/2*a + 2)*q^5 + O(q^6)
sage: chi
Dirichlet character modulo 5 of conductor 5 mapping 2 |--> 1/2*a
sage: f.twist(chi, level=g.level()) == g
True
```

modsym_eigenspace(*sign=0*)

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

EXAMPLES:

```
sage: N = Newform("37a")
sage: N.modular_symbols(0)
Modular Symbols subspace of dimension 2 of Modular Symbols space of dimension 5␣
↪for Gamma_0(37) of weight 2 with sign 0 over Rational Field
sage: M = N.modular_symbols(0)
```

(continues on next page)

(continued from previous page)

```
sage: V = N.modsym_eigenspace(1); V
Vector space of degree 5 and dimension 1 over Rational Field
Basis matrix:
[ 0  1 -1  1  0]
sage: V.0 in M.free_module()
True
sage: V = N.modsym_eigenspace(-1); V
Vector space of degree 5 and dimension 1 over Rational Field
Basis matrix:
[   0    0    0    1 -1/2]
sage: V.0 in M.free_module()
True
```

modular_symbols(*sign=0*)

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

EXAMPLES:

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

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

twist(*chi*, *level=None*, *check=True*)

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

If `self` is a newform f with character ϵ and q-expansion

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

then the twist by χ 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.

- `level` – (optional) the level N of the twisted form. If N is not given, the algorithm tries to compute N using [AL1978], Theorem 3.1; if this is not possible, it returns an error. If N is given but incorrect, i.e. the twisted form does not have level N, then this function will attempt to detect this and return an error, but it may sometimes return an incorrect answer (a newform of level N whose first few coefficients agree with those of $f \otimes \chi$).
- `check` – (optional) boolean; if `True` (default), ensure that the space of modular symbols that is computed is genuinely simple and new. This makes it less likely, but not impossible, that a wrong result is returned if an incorrect `level` is specified.

OUTPUT:

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

EXAMPLES:

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

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

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

AUTHORS:

- Peter Bruin (April 2015)

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

Return the L-series of the modular form Δ.

If algorithm is "gp", this returns an interface to Tim Dokchitser's program for computing with the L-series of the modular form Δ.

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_asymp_coeffs` – integer
- `algorithm` – optional string: 'gp' (default), 'pari'

OUTPUT:

The L-series of Δ.

EXAMPLES:

```
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:

```
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 ε 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
- `eps` - 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:

```
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)
Traceback (most recent call last):
...
ValueError: The given basis vectors must be linearly independent.

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

`sage.modular.modform.hecke_operator_on_qexp.`**`hecke_operator_on_qexp`**(*f*, *n*, *k*, *eps=None*, *prec=None*, *check=True*, *_return_list=False*)

Given the q-expansion f of a modular form with character ε, 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], 3, 12)
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], 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 + 401856*q^14␣
↪+ 1217160*q^15 + 987136*q^16 - 6905934*q^17 + 2727432*q^18 + 10661420*q^19 -␣
↪7109760*q^20 + O(q^21)

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

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

An example on a formal power series:

```
sage: R.<q> = QQ[[]]
sage: f = q + q^2 + q^3 + q^7 + O(q^8)
sage: hecke_operator_on_qexp(f, 3, 12)
```

(continues on next page)

(continued from previous page)

```
q + O(q^3)
sage: hecke_operator_on_qexp(delta_qexp(24), 3, 12).prec()
8
sage: hecke_operator_on_qexp(delta_qexp(25), 3, 12).prec()
9
```

An example of computing $T_{p,k}$ in characteristic p:

```
sage: p = 199
sage: fp = delta_qexp(prec=p^2+1, K=GF(p))
sage: tfp = hecke_operator_on_qexp(fp, p, 12)
sage: tfp == fp[p] * fp
True
sage: tf = hecke_operator_on_qexp(delta_qexp(prec=p^2+1), p, 12).change_ring(GF(p))
sage: tfp == tf
True
```

1.15 Numerical computation of newforms

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

Bases: `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)  # abs tol 1e-12
[3.0, -1.6180339887498947, 0.6180339887498968]
sage: n.systems_of_eigenvalues(7)  # abs tol 2e-12
[
[-1.6180339887498947, 2.2360679774997894, -3.2360679774997894],
[0.6180339887498968, -2.236067977499788, 1.2360679774997936],
[3.0, 4.0, 6.0]
]
sage: n.systems_of_abs(7)  # abs tol 2e-12
[
[0.6180339887498943, 2.2360679774997894, 1.2360679774997887],
[1.6180339887498947, 2.23606797749979, 3.2360679774997894],
[3.0, 4.0, 6.0]
]
sage: n.eigenvalues([2,3,5])  # rel tol 2e-12
[[3.0, -1.6180339887498947, 0.6180339887498968],
 [4.0, 2.2360679774997894, -2.236067977499788],
 [6.0, -3.2360679774997894, 1.2360679774997936]]
```

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:

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

eigenvalues(*primes*)

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

INPUT:

- primes - a list of primes

OUTPUT:

list of lists of eigenvalues.

EXAMPLES:

```
sage: n = numerical_eigenforms(1,12)
sage: n.eigenvalues([3,5,13])  # rel tol 2.4e-10
[[177148.0, 252.00000000001896], [48828126.0, 4830.000000001376],␣
↪[1792160394038.0, -577737.9999898539]]
```

level()

Return the level of this set of modular eigenforms.

EXAMPLES:

```
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:

```
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:

```
sage: numerical_eigenforms(61).systems_of_abs(10)  # rel tol 1e-9
[
[0.3111078174659775, 2.903211925911551, 2.525427560843529, 3.214319743377552],
[1.0, 2.0000000000000027, 3.000000000000003, 1.0000000000000044],
[1.4811943040920152, 0.8060634335253695, 3.1563251746586642, 0.
↪6751308705666477],
[2.170086486626034, 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:

```
sage: numerical_eigenforms(61).systems_of_eigenvalues(10)  # rel tol 1e-9
[
[-1.4811943040920152, 0.8060634335253695, 3.1563251746586642, 0.
↪6751308705666477],
[-1.0, -2.0000000000000027, -3.000000000000003, 1.0000000000000044],
[0.3111078174659775, 2.903211925911551, -2.525427560843529, -3.214319743377552],
[2.170086486626034, -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:

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

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

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

EXAMPLES:

```
sage: sage.modular.modform.numerical.support( numerical_eigenforms(61)._easy_
↪vector(), 1.0 )
[]

sage: sage.modular.modform.numerical.support( numerical_eigenforms(61)._easy_
↪vector(), 0.5 )
[0, 4]
```

1.16 The Victor Miller Basis

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

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

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

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

INPUT:

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

OUTPUT:

a power series over K in the variable `var`

ALGORITHM:

Compute the theta series

$$\sum_{n \geq 0} (-1)^n (2n+1) q^{n(n+1)/2},$$

a very simple explicit modular form whose 8th power is Δ. Then compute the 8th power. All computations are done over $\mathbf{Z}$ or $\mathbf{Z}$ modulo N depending on the characteristic of the given coefficient ring K, and coerced into K afterwards.

EXAMPLES:

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

AUTHORS:

- William Stein: original code
- David Harvey (2007-05): sped up first squaring step
- Martin Raum (2009-08-02): use FLINT for polynomial arithmetic (instead of NTL)

sage.modular.modform.vm_basis.**victor_miller_basis**(*k*, *prec=10*, *cusp_only=False*, *var='q'*)

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

INPUT:

- `k` – an integer
- `prec` – (default: 10) a positive integer
- `cusp_only` – bool (default: False)
- `var` – string (default: 'q')

OUTPUT:

A sequence whose entries are power series in `ZZ[[var]]`.

EXAMPLES:

```
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)
```

(continues on next page)

(continued from previous page)

```
]

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)

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 χ. The modulus of χ 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)
[q - 2*q^2 - q^9 + 2*q^14 + 6*q^18 - 2*q^21 - 4*q^22 - q^25 + O(q^30),
 q^2 - q^14 - 3*q^18 + 2*q^22 + O(q^30),
 q^4 - q^8 - q^16 + q^28 + O(q^30),
 q^7 - 2*q^15 + O(q^30)]
```

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

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

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

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

(continues on next page)

(continued from previous page)

```
 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)
[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)
[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)]
```

ALGORITHM: Basmaji (page 55 of his Essen thesis, "Ein Algorithmus zur Berechnung von Hecke-Operatoren und Anwendungen auf modulare Kurven", http://wstein.org/scans/papers/basmaji/).

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

1.18 Graded rings of modular forms

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

AUTHORS:

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

class `sage.modular.modform.ring.`**`ModularFormsRing`**(*group*, *base_ring=Rational Field*)

Bases: `sage.structure.parent.Parent`

The ring of modular forms (of weights 0 or at least 2) for a congruence subgroup of $\mathrm{SL}_2(\mathbf{Z})$, with coefficients in a specified base ring.

EXAMPLES:

```
sage: ModularFormsRing(Gamma1(13))
Ring of Modular Forms for Congruence Subgroup Gamma1(13) over Rational Field
sage: m = ModularFormsRing(4); m
Ring of Modular Forms for Congruence Subgroup Gamma0(4) over Rational Field
sage: m.modular_forms_of_weight(2)
Modular Forms space of dimension 2 for Congruence Subgroup Gamma0(4) of weight 2␣
↪over Rational Field
sage: m.modular_forms_of_weight(10)
Modular Forms space of dimension 6 for Congruence Subgroup Gamma0(4) of weight 10␣
↪over Rational Field
sage: m == loads(dumps(m))
True
```

(continues on next page)

(continued from previous page)

```
sage: m.generators()
[(2, 1 + 24*q^2 + 24*q^4 + 96*q^6 + 24*q^8 + O(q^10)),
 (2, q + 4*q^3 + 6*q^5 + 8*q^7 + 13*q^9 + O(q^10))]
sage: m.q_expansion_basis(2,10)
[1 + 24*q^2 + 24*q^4 + 96*q^6 + 24*q^8 + O(q^10),
 q + 4*q^3 + 6*q^5 + 8*q^7 + 13*q^9 + O(q^10)]
sage: m.q_expansion_basis(3,10)
[]
sage: m.q_expansion_basis(10,10)
[1 + 10560*q^6 + 3960*q^8 + O(q^10),
 q - 8056*q^7 - 30855*q^9 + O(q^10),
 q^2 - 796*q^6 - 8192*q^8 + O(q^10),
 q^3 + 66*q^7 + 832*q^9 + O(q^10),
 q^4 + 40*q^6 + 528*q^8 + O(q^10),
 q^5 + 20*q^7 + 190*q^9 + O(q^10)]
```

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

```
sage: M = ModularFormsRing(1)
sage: M.ngens()
2
sage: E4, E6 = polygens(QQ, 'E4, E6')
sage: M(E4)
1 + 240*q + 2160*q^2 + 6720*q^3 + 17520*q^4 + 30240*q^5 + O(q^6)
sage: M(E6)
1 - 504*q - 16632*q^2 - 122976*q^3 - 532728*q^4 - 1575504*q^5 + O(q^6)
sage: M((E4^3 - E6^2)/1728)
q - 24*q^2 + 252*q^3 - 1472*q^4 + 4830*q^5 + O(q^6)
```

Element

alias of *sage.modular.modform.element.GradedModularFormElement*

change_ring(*base_ring*)

Return the ring of modular forms over the given base ring and the same group as `self`.

INPUT:

- `base_ring` – a base ring, which should be $\mathbf{Q}$, $\mathbf{Z}$, or the integers mod p for some prime p.

EXAMPLES:

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

cuspidal_ideal_generators(*maxweight=8*, *prec=None*)

Calculate generators for the ideal of cuspidal forms in this ring, as a module over the whole ring.

EXAMPLES:

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

cuspidal_submodule_q_expansion_basis(*weight*, *prec=None*)

Calculate a basis of q-expansions for the space of cusp forms of weight `weight` for this group.

INPUT:

- `weight` (integer) – the weight
- `prec` (integer or None) – precision of q-expansions to return

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

EXAMPLES:

```
sage: R = ModularFormsRing(Gamma0(3))
sage: R.cuspidal_submodule_q_expansion_basis(20)
[q - 8532*q^6 - 88442*q^7 + O(q^8), q^2 + 207*q^6 + 24516*q^7 + 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
```

from_polynomial(*polynomial*, *gens=None*)

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

INPUT:

- `polynomial` – A multivariate polynomial. The variables names of the polynomial should be different from `'q'`. The number of variable of this polynomial should equal the number of generators
- `gens` – list (default: `None`) of generators of the modular forms ring

OUTPUT: A `GradedModularFormElement` given by the polynomial relation `polynomial`.

EXAMPLES:

```
sage: M = ModularFormsRing(1)
sage: x,y = polygens(QQ, 'x,y')
sage: M.from_polynomial(x^2+y^3)
2 - 1032*q + 774072*q^2 - 77047584*q^3 - 11466304584*q^4 - 498052467504*q^5 +␣
↪O(q^6)
```

(continues on next page)

(continued from previous page)

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

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

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

gen(*i*)

Return the i-th generator of `self`.

INPUT:

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

OUTPUT: A `GradedModularFormElement`

EXAMPLES:

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

gen_forms(*maxweight=8*, *start_gens=[]*, *start_weight=2*)

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

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

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

generators(*maxweight=8*, *prec=10*, *start_gens=[]*, *start_weight=2*)

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

INPUT:

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

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

OUTPUT:

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

See also:

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

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

EXAMPLES:

```
sage: ModularFormsRing(SL2Z).generators()
[(4, 1 + 240*q + 2160*q^2 + 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
```

(continues on next page)

(continued from previous page)

```
[(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 + 2160*q^2 + 6720*q^3 + 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 + 96*q^3 + 24*q^4 + 144*q^5 + 96*q^6 + 192*q^7 + 24*q^8␣
↪+ 312*q^9 + 144*q^10 + 288*q^11 + O(q^12)),
 (4, 1 + 240*q^2 + 2160*q^4 + 6720*q^6 + 17520*q^8 + 30240*q^10 + O(q^12))]
sage: ModularFormsRing(4).generators(maxweight=2, prec=20)
[(2, 1 + 24*q^2 + 24*q^4 + 96*q^6 + 24*q^8 + 144*q^10 + 96*q^12 + 192*q^14 +␣
↪24*q^16 + 312*q^18 + O(q^20)),
 (2, q + 4*q^3 + 6*q^5 + 8*q^7 + 13*q^9 + 12*q^11 + 14*q^13 + 24*q^15 + 18*q^17␣
↪+ 20*q^19 + 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: from sage.modular.dims import dimension_modular_forms
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 ≥ 2:

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

Using different base rings will change the generators:

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

(continues on next page)

(continued from previous page)

```
 (4, q^2 + O(q^4)),
 (4, -2*q^3 + O(q^4)),
 (6, 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]
sage: [k for k,f in ModularFormsRing(1, ZZ).generators(maxweight=12)]
[4, 6, 12]
sage: [k for k,f in ModularFormsRing(1, Zmod(5)).generators(maxweight=12)]
[4, 6]
sage: [k for k,f in ModularFormsRing(1, Zmod(2)).generators(maxweight=12)]
[4, 6, 12]
```

An example where `start_gens` are specified:

```
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))]
```

gens(*maxweight=8*, *start_gens=[]*, *start_weight=2*)

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

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

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

group()

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

EXAMPLES:

```
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:

```
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
```

ngens()

Return the number of generators of `self`

EXAMPLES:

```
sage: ModularFormsRing(1).ngens()
2
sage: ModularFormsRing(Gamma0(2)).ngens()
2
sage: ModularFormsRing(Gamma1(13)).ngens() # long time
33
```

Warning: Computing the number of generators of a graded ring of modular form for a certain congruence subgroup can be very long.

one()

Return the one element of this ring.

EXAMPLES:

```
sage: M = ModularFormsRing(1)
sage: u = M.one(); u
1
sage: u.is_one()
```

(continues on next page)

(continued from previous page)

```
True
sage: u + u
2
sage: E4 = ModularForms(1,4).0; E4
1 + 240*q + 2160*q^2 + 6720*q^3 + 17520*q^4 + 30240*q^5 + O(q^6)
sage: E4 * u
1 + 240*q + 2160*q^2 + 6720*q^3 + 17520*q^4 + 30240*q^5 + O(q^6)
```

polynomial_ring(*names*, *gens=None*)

Return a polynomial ring of which `self` is a quotient.

INPUT:

- `names` – a list or tuple of names (strings), or a comma separated string
- `gens` (default: None) – (list) a list of generator of `self`. If `gens` is `None` then the generators returned by `gen_forms()` is used instead.

OUTPUT: A multivariate polynomial ring in the variable `names`. Each variable of the polynomial ring correspond to a generator given in gens (following the ordering of the list).

EXAMPLES:

```
sage: M = ModularFormsRing(1)
sage: gens = M.gen_forms()
sage: M.polynomial_ring('E4, E6', gens)
Multivariate Polynomial Ring in E4, E6 over Rational Field
sage: M = ModularFormsRing(Gamma0(8))
sage: gens = M.gen_forms()
sage: M.polynomial_ring('g', gens)
Multivariate Polynomial Ring in g0, g1, g2 over Rational Field
```

The degrees of the variables are the weights of the corresponding forms:

```
sage: M = ModularFormsRing(1)
sage: P.<E4, E6> = M.polynomial_ring()
sage: E4.degree()
4
sage: E6.degree()
6
sage: (E4*E6).degree()
10
```

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 + O(q^10),
 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 + O(q^10)]
```

We calculate a basis of a massive modular forms space, in two ways. Using this module is about twice as fast as Sage's generic code.

```
sage: A = ModularFormsRing(11).q_expansion_basis(30, prec=40) # long time (5s)
sage: B = ModularForms(Gamma0(11), 30).q_echelon_basis(prec=40) # long time (9s)
sage: A == B # long time
True
```

Check that absurdly small values of `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)]
```

some_elements()

Return a list of generators of `self`.

EXAMPLES:

```
sage: ModularFormsRing(1).some_elements()
[1 + 240*q + 2160*q^2 + 6720*q^3 + 17520*q^4 + 30240*q^5 + O(q^6),
 1 - 504*q - 16632*q^2 - 122976*q^3 - 532728*q^4 - 1575504*q^5 + O(q^6)]
```

zero()

Return the zero element of this ring.

EXAMPLES:

```
sage: M = ModularFormsRing(1)
sage: zer = M.zero(); zer
0
sage: zer.is_zero()
True
sage: E4 = ModularForms(1,4).0; E4
1 + 240*q + 2160*q^2 + 6720*q^3 + 17520*q^4 + 30240*q^5 + O(q^6)
sage: E4 + zer
1 + 240*q + 2160*q^2 + 6720*q^3 + 17520*q^4 + 30240*q^5 + O(q^6)
sage: zer * E4
0
sage: E4 * zer
0
```

1.19 q-expansion of j-invariant

sage.modular.modform.j_invariant.**j_invariant_qexp**(*prec=10*, *K=Rational Field*)

Return the q-expansion of the j-invariant to precision `prec` in the field K.

See also:

If you want to evaluate (numerically) the j-invariant at certain points, see the special function `elliptic_j()`.

Warning: Stupid algorithm – we divide by Delta, which is slow.

EXAMPLES:

```
sage: j_invariant_qexp(4)
q^-1 + 744 + 196884*q + 21493760*q^2 + 864299970*q^3 + O(q^4)
sage: j_invariant_qexp(2)
q^-1 + 744 + 196884*q + O(q^2)
sage: j_invariant_qexp(100, GF(2))
q^-1 + q^7 + q^15 + q^31 + q^47 + q^55 + q^71 + q^87 + O(q^100)
```

1.20 q-expansions of Theta Series

AUTHOR:

William Stein

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

Return the q-expansion of the series ` theta_2 = sum_{n 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:

```
sage: theta2_qexp(18)
q + q^9 + O(q^18)
sage: theta2_qexp(49)
q + q^9 + q^25 + O(q^49)
sage: theta2_qexp(100, 'q', QQ)
q + q^9 + q^25 + q^49 + q^81 + O(q^100)
sage: f = theta2_qexp(100, 't', GF(3)); f
t + t^9 + t^25 + t^49 + t^81 + O(t^100)
sage: parent(f)
Power Series Ring in t over Finite Field of size 3
sage: theta2_qexp(200)
q + q^9 + q^25 + q^49 + q^81 + q^121 + q^169 + O(q^200)
```

(continues on next page)

(continued from previous page)

```
sage: f = theta2_qexp(20,sparse=True); f
q + q^9 + O(q^20)
sage: parent(f)
Sparse Power Series Ring in q over Integer Ring
```

sage.modular.modform.theta.**theta_qexp**(*prec=10*, *var='q'*, *K=Integer Ring*, *sparse=False*)

Return the q-expansion of the standard θ series ` theta = 1 + 2sum_{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 + 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 + O(t^
↪100)
sage: theta_qexp(100, 't', GF(2))
1 + O(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
```

CHAPTER

TWO

DESIGN NOTES

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

PYTHON MODULE INDEX

m

sage.modular.modform.ambient, 21
sage.modular.modform.ambient_eps, 28
sage.modular.modform.ambient_g0, 31
sage.modular.modform.ambient_g1, 31
sage.modular.modform.ambient_R, 33
sage.modular.modform.constructor, 1
sage.modular.modform.cuspidal_submodule, 34
sage.modular.modform.eis_series, 44
sage.modular.modform.eis_series_cython, 47
sage.modular.modform.eisenstein_submodule, 38
sage.modular.modform.element, 47
sage.modular.modform.half_integral, 84
sage.modular.modform.hecke_operator_on_qexp, 76
sage.modular.modform.j_invariant, 95
sage.modular.modform.notes, 97
sage.modular.modform.numerical, 78
sage.modular.modform.ring, 85
sage.modular.modform.space, 6
sage.modular.modform.submodule, 34
sage.modular.modform.theta, 95
sage.modular.modform.vm_basis, 81

INDEX

A

abelian_variety() (sage.modular.modform.element.Newform method), 68
ambient_space() (sage.modular.modform.ambient.ModularFormsAmbient method), 22
ap() (sage.modular.modform.numerical.NumericalEigenforms method), 79
atkin_lehner_action() (sage.modular.modform.element.Newform method), 69
atkin_lehner_eigenvalue() (sage.modular.modform.element.ModularForm_abstract method), 58
atkin_lehner_eigenvalue() (sage.modular.modform.element.ModularFormElement method), 55
atkin_lehner_eigenvalue() (sage.modular.modform.element.ModularFormElement_elliptic_curve method), 57
atkin_lehner_eigenvalue() (sage.modular.modform.element.Newform method), 70

B

basis() (sage.modular.modform.space.ModularFormsSpace method), 7

C

canonical_parameters() (in module sage.modular.modform.constructor), 5
change_ring() (sage.modular.modform.ambient.ModularFormsAmbient method), 22
change_ring() (sage.modular.modform.ambient_eps.ModularFormsAmbient_eps method), 29
change_ring() (sage.modular.modform.ambient_R.ModularFormsAmbient_R method), 33
change_ring() (sage.modular.modform.cuspidal_submodule.CuspidalSubmodule method), 35
change_ring() (sage.modular.modform.eisenstein_submodule.EisensteinSubmodule_params method), 40
change_ring() (sage.modular.modform.ring.ModularFormsRing method), 86
character() (sage.modular.modform.element.EisensteinSeries method), 48
character() (sage.modular.modform.element.ModularForm_abstract method), 58
character() (sage.modular.modform.element.Newform method), 71
character() (sage.modular.modform.space.ModularFormsSpace method), 7
chi() (sage.modular.modform.element.EisensteinSeries method), 49
cm_discriminant() (sage.modular.modform.element.ModularForm_abstr method), 59
coefficient() (sage.modular.modform.element.ModularForm_abstract method), 59
coefficient() (sage.modular.modform.element.Newform method), 71
coefficients() (sage.modular.modform.element.ModularForm_abstract method), 59
compute_eisenstein_params() (in module sage.modular.modform.eis_series), 44
contains_each() (in module sage.modular.modform.space), 21
CuspForms() (in module sage.modular.modform.constructor), 1
cuspidal_ideal_generators() (sage.modular.modform.ring.ModularFormsRing method), 86
cuspidal_submodule() (sage.modular.modform.ambient.ModularFormsAmbient method), 23
cuspidal_submodule() (sage.modular.modform.ambient_eps.ModularFormsAmbient_eps method), 29
cuspidal_submodule() (sage.modular.modform.ambient_g0.ModularFormsAmbient_g0_ method), 31
cuspidal_submodule() (sage.modular.modform.ambient_g1.ModularFormsAmbient_g1_ method), 32
cuspidal_submodule() (sage.modular.modform.ambient_g1.ModularFormsAmbient_gH_ method), 32

cuspidal_submodule() (sage.modular.modform.ambient_R.ModularFormsAmbient_R method), 33
cuspidal_submodule() (sage.modular.modform.space.ModularFormsSpace method), 8
cuspidal_submodule_q_expansion_basis() (sage.modular.modform.ring.ModularFormsRing method), 87
cuspidal_subspace() (sage.modular.modform.space.ModularFormsSpace method), 8
CuspidalSubmodule (class in sage.modular.modform.cuspidal_submodule), 35
CuspidalSubmodule_eps (class in sage.modular.modform.cuspidal_submodule), 36
CuspidalSubmodule_g0_Q (class in sage.modular.modform.cuspidal_submodule), 37
CuspidalSubmodule_g1_Q (class in sage.modular.modform.cuspidal_submodule), 37
CuspidalSubmodule_gH_Q (class in sage.modular.modform.cuspidal_submodule), 37
CuspidalSubmodule_level1_Q (class in sage.modular.modform.cuspidal_submodule), 37
CuspidalSubmodule_modsym_qexp (class in sage.modular.modform.cuspidal_submodule), 37
CuspidalSubmodule_R (class in sage.modular.modform.cuspidal_submodule), 36
CuspidalSubmodule_wt1_eps (class in sage.modular.modform.cuspidal_submodule), 38
CuspidalSubmodule_wt1_gH (class in sage.modular.modform.cuspidal_submodule), 38
cyclotomic_restriction() (in module sage.modular.modform.eisenstein_submodule), 43
cyclotomic_restriction_tower() (in module sage.modular.modform.eisenstein_submodule), 43

D

decomposition() (sage.modular.modform.space.ModularFormsSpace method), 9
delta_lseries() (in module sage.modular.modform.element), 75
delta_qexp() (in module sage.modular.modform.vm_basis), 81
derivative() (sage.modular.modform.element.GradedModularFormElem method), 50
dimension() (sage.modular.modform.ambient.ModularFormsAmbient method), 23

E

echelon_basis() (sage.modular.modform.space.ModularFormsSpace method), 9
echelon_form() (sage.modular.modform.space.ModularFormsSpace method), 10
eigenvalues() (sage.modular.modform.numerical.NumericalEigenforms method), 79
eisenstein_params() (sage.modular.modform.ambient.ModularFormsAmbient method), 23
eisenstein_series() (sage.modular.modform.ambient.ModularFormsAmbient method), 23
eisenstein_series() (sage.modular.modform.eisenstein_submodule.EisensteinSubmodu method), 40
eisenstein_series() (sage.modular.modform.space.ModularFormsSpace method), 11
eisenstein_series_lseries() (in module sage.modular.modform.eis_series), 45
eisenstein_series_poly() (in module sage.modular.modform.eis_series_cython), 47
eisenstein_series_qexp() (in module sage.modular.modform.eis_series), 45
eisenstein_submodule() (sage.modular.modform.ambient.ModularFormsAmbient method), 24
eisenstein_submodule() (sage.modular.modform.ambient_eps.ModularFormsAmbient_eps method), 29
eisenstein_submodule() (sage.modular.modform.ambient_g0.ModularFormsAmbient_g0_ method), 31
eisenstein_submodule() (sage.modular.modform.ambient_g1.ModularFormsAmbient_g1_ method), 32
eisenstein_submodule() (sage.modular.modform.ambient_g1.ModularFormsAmbient_gH_ method), 32
eisenstein_submodule() (sage.modular.modform.eisenstein_submodule.EisensteinSubmodu method), 38
eisenstein_submodule() (sage.modular.modform.space.ModularFormsSpace method), 11

eisenstein_subspace() (*sage.modular.modform.space.ModularFormsSpace method*), 11
EisensteinForms() (*in module sage.modular.modform.constructor*), 1
EisensteinSeries (*class in sage.modular.modform.element*), 47
EisensteinSubmodule (*class in sage.modular.modform.eisenstein_submodule*), 38
EisensteinSubmodule_eps (*class in sage.modular.modform.eisenstein_submodule*), 39
EisensteinSubmodule_g0_Q (*class in sage.modular.modform.eisenstein_submodule*), 39
EisensteinSubmodule_g1_Q (*class in sage.modular.modform.eisenstein_submodule*), 40
EisensteinSubmodule_gH_Q (*class in sage.modular.modform.eisenstein_submodule*), 40
EisensteinSubmodule_params (*class in sage.modular.modform.eisenstein_submodule*), 40
Ek_ZZ() (*in module sage.modular.modform.eis_series_cython*), 47
Element (*sage.modular.modform.ring.ModularFormsRing attribute*), 86
Element (*sage.modular.modform.space.ModularFormsSpace attribute*), 7
element() (*sage.modular.modform.element.Newform method*), 72
elliptic_curve() (*sage.modular.modform.element.ModularFormElement_elliptic_curve method*), 58
embedded_submodule() (*sage.modular.modform.space.ModularFormsSpace method*), 11

F

find_in_space() (*sage.modular.modform.space.ModularFormsSpace method*), 12
free_module() (*sage.modular.modform.ambient.ModularFormsAmbient method*), 24
from_polynomial() (*sage.modular.modform.ring.ModularFormsRing method*), 87

G

gen() (*sage.modular.modform.ring.ModularFormsRing method*), 88
gen() (*sage.modular.modform.space.ModularFormsSpace method*), 12
gen_forms() (*sage.modular.modform.ring.ModularFormsRing method*), 88
generators() (*sage.modular.modform.ring.ModularFormsRing method*), 89
gens() (*sage.modular.modform.ring.ModularFormsRing method*), 91
gens() (*sage.modular.modform.space.ModularFormsSpace method*), 13
GradedModularFormElement (*class in sage.modular.modform.element*), 50
group() (*sage.modular.modform.element.GradedModularFormElement method*), 51
group() (*sage.modular.modform.element.ModularForm_abstract method*), 59
group() (*sage.modular.modform.ring.ModularFormsRing method*), 92
group() (*sage.modular.modform.space.ModularFormsSpace method*), 13

H

half_integral_weight_modform_basis() (*in module sage.modular.modform.half_integral*), 84
has_character() (*sage.modular.modform.space.ModularFormsSpace method*), 13
has_cm() (*sage.modular.modform.element.ModularForm_abstract method*), 60
hecke_eigenvalue_field() (*sage.modular.modform.element.Newform method*), 72
hecke_module_of_level() (*sage.modular.modform.ambient.ModularFormsAmbient method*), 24
hecke_module_of_level() (*sage.modular.modform.ambient_eps.ModularFormsAmbient_eps method*), 30
hecke_operator_on_basis() (*in module sage.modular.modform.hecke_operator_on_qexp*), 76
hecke_operator_on_qexp() (*in module sage.modular.modform.hecke_operator_on_qexp*), 77
hecke_polynomial() (*sage.modular.modform.ambient.ModularFormsAm method*), 25
hecke_polynomial() (*sage.modular.modform.cuspidal_submodule.Cuspi method*), 37
homogeneous_component() (*sage.modular.modform.element.GradedModularFormElement method*), 51

I

integral_basis() (*sage.modular.modform.space.ModularFormsSpace method*), 14
is_ambient() (*sage.modular.modform.ambient.ModularFormsAmbient method*), 25
is_ambient() (*sage.modular.modform.space.ModularFormsSpace method*), 14

is_cuspidal() (sage.modular.modform.cuspidal_submodule.CuspidalSubmodule method), 35
is_cuspidal() (sage.modular.modform.element.Newform method), 73
is_cuspidal() (sage.modular.modform.space.ModularFormsSpace method), 15
is_eisenstein() (sage.modular.modform.space.ModularFormsSpace method), 15
is_homogeneous() (sage.modular.modform.element.GradedModularFormElement method), 51
is_homogeneous() (sage.modular.modform.element.ModularForm_abstract method), 60
is_modular_form() (sage.modular.modform.element.GradedModularFormElement method), 52
is_modular_form() (sage.modular.modform.element.ModularForm_abstract method), 61
is_ModularFormElement() (in module sage.modular.modform.element), 76
is_ModularFormsSpace() (in module sage.modular.modform.space), 21
is_one() (sage.modular.modform.element.GradedModularFormElement method), 52
is_zero() (sage.modular.modform.element.GradedModularFormElement method), 52

J

j_invariant_qexp() (in module sage.modular.modform.j_invariant), 95

L

L() (sage.modular.modform.element.EisensteinSeries method), 48
level() (sage.modular.modform.element.ModularForm_abstract method), 61
level() (sage.modular.modform.numerical.NumericalEigenforms method), 80
level() (sage.modular.modform.space.ModularFormsSpace method), 15
local_component() (sage.modular.modform.element.Newform method), 73
lseries() (sage.modular.modform.element.ModularForm_abstract method), 61

M

M() (sage.modular.modform.element.EisensteinSeries method), 48
minimal_twist() (sage.modular.modform.element.Newform method), 73
modsym_eigenspace() (sage.modular.modform.element.Newform method), 73
modular_forms_of_weight() (sage.modular.modform.ring.ModularFormsRing method), 92
modular_symbols() (sage.modular.modform.ambient.ModularFormsAmb method), 25
modular_symbols() (sage.modular.modform.ambient_eps.ModularForms method), 30
modular_symbols() (sage.modular.modform.ambient_R.ModularFormsA method), 33
modular_symbols() (sage.modular.modform.cuspidal_submodule.Cuspid method), 35
modular_symbols() (sage.modular.modform.eisenstein_submodule.Eisen method), 38
modular_symbols() (sage.modular.modform.element.Newform method), 74
modular_symbols() (sage.modular.modform.numerical.NumericalEigenfo method), 80
modular_symbols() (sage.modular.modform.space.ModularFormsSpace method), 15
ModularForm_abstract (class in sage.modular.modform.element), 58
ModularFormElement (class in sage.modular.modform.element), 55
ModularFormElement_elliptic_curve (class in sage.modular.modform.element), 57
ModularForms() (in module sage.modular.modform.constructor), 1
ModularForms_clear_cache() (in module sage.modular.modform.constructor), 4
ModularFormsAmbient (class in sage.modular.modform.ambient), 22
ModularFormsAmbient_eps (class in sage.modular.modform.ambient_eps), 29
ModularFormsAmbient_g0_Q (class in sage.modular.modform.ambient_g0), 31
ModularFormsAmbient_g1_Q (class in sage.modular.modform.ambient_g1), 32
ModularFormsAmbient_gH_Q (class in sage.modular.modform.ambient_g1), 32
ModularFormsAmbient_R (class in sage.modular.modform.ambient_R), 33
ModularFormsRing (class in sage.modular.modform.ring), 85
ModularFormsSpace (class in sage.modular.modform.space), 7
ModularFormsSubmodule (class in sage.modular.modform.submodule), 34
ModularFormsSubmoduleWithBasis (class in sage.modular.modform.submodule), 34
module
  sage.modular.modform.ambient, 21
  sage.modular.modform.ambient_eps, 28
  sage.modular.modform.ambient_g0, 31
  sage.modular.modform.ambient_g1, 31
  sage.modular.modform.ambient_R, 33
  sage.modular.modform.constructor, 1
  sage.modular.modform.cuspidal_submodule,

34
sage.modular.modform.eis_series, 44
sage.modular.modform.eis_series_cython, 47
sage.modular.modform.eisenstein_submodule, 38
sage.modular.modform.element, 47
sage.modular.modform.half_integral, 84
sage.modular.modform.hecke_operator_on_qexp, 76
sage.modular.modform.j_invariant, 95
sage.modular.modform.notes, 97
sage.modular.modform.numerical, 78
sage.modular.modform.ring, 85
sage.modular.modform.space, 6
sage.modular.modform.submodule, 34
sage.modular.modform.theta, 95
sage.modular.modform.vm_basis, 81
module() (*sage.modular.modform.ambient.ModularFormsAmbient method*), 25

N

new_eisenstein_series() (*sage.modular.modform.eisenstein_submodule.EisensteinSubmodule_params method*), 42
new_level() (*sage.modular.modform.element.EisensteinSeries method*), 49
new_submodule() (*sage.modular.modform.ambient.ModularFormsAmbient method*), 26
new_submodule() (*sage.modular.modform.cuspidal_submodule.CuspidalSubmodule_modsym_qexp method*), 37
new_submodule() (*sage.modular.modform.eisenstein_submodule.EisensteinSubmodule_params method*), 42
new_submodule() (*sage.modular.modform.space.ModularFormsSpace method*), 15
new_subspace() (*sage.modular.modform.space.ModularFormsSpace method*), 16
Newform (*class in sage.modular.modform.element*), 68
Newform() (*in module sage.modular.modform.constructor*), 4
Newforms() (*in module sage.modular.modform.constructor*), 5
newforms() (*sage.modular.modform.space.ModularFormsSpace method*), 16
ngens() (*sage.modular.modform.ring.ModularFormsRing method*), 92
number() (*sage.modular.modform.element.Newform method*), 74
NumericalEigenforms (*class in sage.modular.modform.numerical*), 78

O

one() (*sage.modular.modform.ring.ModularFormsRing method*), 92

P

padded_list() (*sage.modular.modform.element.ModularForm_abstract method*), 63
parameters() (*sage.modular.modform.eisenstein_submodule.EisensteinSu method*), 42
parameters() (*sage.modular.modform.element.EisensteinSeries method*), 49
parse_label() (*in module sage.modular.modform.constructor*), 6
period() (*sage.modular.modform.element.ModularForm_abstract method*), 63
petersson_norm() (*sage.modular.modform.element.ModularForm_abstra method*), 65
polynomial_ring() (*sage.modular.modform.ring.ModularFormsRing method*), 93
prec() (*sage.modular.modform.ambient.ModularFormsAmbient method*), 26
prec() (*sage.modular.modform.element.ModularForm_abstract method*), 65
prec() (*sage.modular.modform.space.ModularFormsSpace method*), 16
psi() (*sage.modular.modform.element.EisensteinSeries method*), 49

Q

q_echelon_basis() (*sage.modular.modform.space.ModularFormsSpace method*), 17
q_expansion() (*sage.modular.modform.element.GradedModularFormEle method*), 52
q_expansion() (*sage.modular.modform.element.ModularForm_abstract method*), 65
q_expansion_basis() (*sage.modular.modform.ring.ModularFormsRing method*), 93
q_expansion_basis() (*sage.modular.modform.space.ModularFormsSpace method*), 17
q_integral_basis() (*sage.modular.modform.space.ModularFormsSpace method*), 18
qexp() (*sage.modular.modform.element.GradedModularFormElement method*), 53
qexp() (*sage.modular.modform.element.ModularForm_abstract method*), 66

R

rank() (*sage.modular.modform.ambient.ModularFormsAmbient method*), 27

S

sage.modular.modform.ambient
module, 21
sage.modular.modform.ambient_eps
module, 28

sage.modular.modform.ambient_g0
module, 31
sage.modular.modform.ambient_g1
module, 31
sage.modular.modform.ambient_R
module, 33
sage.modular.modform.constructor
module, 1
sage.modular.modform.cuspidal_submodule
module, 34
sage.modular.modform.eis_series
module, 44
sage.modular.modform.eis_series_cython
module, 47
sage.modular.modform.eisenstein_submodule
module, 38
sage.modular.modform.element
module, 47
sage.modular.modform.half_integral
module, 84
sage.modular.modform.hecke_operator_on_qexp
module, 76
sage.modular.modform.j_invariant
module, 95
sage.modular.modform.notes
module, 97
sage.modular.modform.numerical
module, 78
sage.modular.modform.ring
module, 85
sage.modular.modform.space
module, 6
sage.modular.modform.submodule
module, 34
sage.modular.modform.theta
module, 95
sage.modular.modform.vm_basis
module, 81
serre_derivative() (*sage.modular.modform.element.GradedModularFormElement method*), 53
serre_derivative() (*sage.modular.modform.element.ModularForm_abstract method*), 66
set_precision() (*sage.modular.modform.ambient.ModularFormsAmbient method*), 27
set_precision() (*sage.modular.modform.space.ModularFormsSpace method*), 18
some_elements() (*sage.modular.modform.ring.ModularFormsRing method*), 94
span() (*sage.modular.modform.space.ModularFormsSpace method*), 19
span_of_basis() (*sage.modular.modform.space.ModularFormsSpace method*), 19
sturm_bound() (*sage.modular.modform.space.ModularFormsSpace method*), 20
support() (*in module sage.modular.modform.numerical*), 81
symsquare_lseries() (*sage.modular.modform.element.ModularForm_abstract method*), 67
systems_of_abs() (*sage.modular.modform.numerical.NumericalEigenfor method*), 80
systems_of_eigenvalues() (*sage.modular.modform.numerical.NumericalEigenforms method*), 80

T

t() (*sage.modular.modform.element.EisensteinSeries method*), 49
theta2_qexp() (*in module sage.modular.modform.theta*), 95
theta_qexp() (*in module sage.modular.modform.theta*), 96
to_polynomial() (*sage.modular.modform.element.GradedModularFormE method*), 54
twist() (*sage.modular.modform.element.ModularFormElement method*), 56
twist() (*sage.modular.modform.element.Newform method*), 74

V

valuation() (*sage.modular.modform.element.ModularForm_abstract method*), 68
victor_miller_basis() (*in module sage.modular.modform.vm_basis*), 82

W

weight() (*sage.modular.modform.element.GradedModularFormElement method*), 54
weight() (*sage.modular.modform.element.ModularForm_abstract method*), 68
weight() (*sage.modular.modform.numerical.NumericalEigenforms method*), 80
weight() (*sage.modular.modform.space.ModularFormsSpace method*), 21
weights_list() (*sage.modular.modform.element.GradedModularFormEl method*), 55

Z

zero() (*sage.modular.modform.ring.ModularFormsRing method*), 94