Sage Reference Manual: Modular Forms for Hecke Triangle Groups

Release 7.5

The Sage Development Team

Jan 12, 2017

CHAPTER

ONE

OVERVIEW OF HECKE TRIANGLE GROUPS AND MODULAR FORMS FOR HECKE TRIANGLE GROUPS

AUTHORS:

- Jonas Jermann (2013): initial version

1.1 Hecke triangle groups and elements:

- **Hecke triangle group:** The Von Dyck group corresponding to the triangle group with angles `(pi/2,pi/n,0)` for `n=3,4,5,...`, generated by the conformal circle inversion `S` and by the translation `T` by `lambda=2*cos(pi/n)`. I.e. the subgroup of orientation preserving elements of the triangle group generated by reflections along the boundaries of the above hyperbolic triangle. The group is arithmetic iff `n=3,4,6,infinity`.

  The group elements correspond to matrices over ZZ[lambda], namely the corresponding order in the number field defined by the minimal polynomial of lambda (which embedds into `AlgebraicReal` accordingly).

  An exact symbolic expression of the corresponding transfinite diameter `d` (which is used as a formal parameter for Fourier expansion of modular forms) can be obtained. For arithmetic groups the (correct) rational number is returned instead.

  Basic matrices like `S,T,U,V(j)` are available.

  EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.hecke_triangle_groups import␣
↪HeckeTriangleGroup
sage: G = HeckeTriangleGroup(12)
sage: G
Hecke triangle group for n = 12
sage: G.is_arithmetic()
False
sage: G.dvalue()
e^(2*euler_gamma - 4*pi/(sqrt(6) + sqrt(2)) + psi(19/24) + psi(17/24))
sage: AA(G.lam())
1.9318516525781...?

sage: G = HeckeTriangleGroup(6)
sage: G
Hecke triangle group for n = 6
sage: G.is_arithmetic()
True
sage: G.dvalue()
1/108
```

```
sage: AA(G.lam()) == AA(sqrt(3))
True
sage: G.gens()
(
[ 0 -1]  [  1 lam]
[ 1  0], [  0   1]
)
sage: G.U()^3
[ lam   -2]
[   2 -lam]
sage: G.U().parent()
Hecke triangle group for n = 6
sage: G.U().matrix().parent()
Full MatrixSpace of 2 by 2 dense matrices over Maximal Order in Number Field in␣
↪lam with defining polynomial x^2 - 3
```

- **Decomposition into product of generators:** It is possible to decompose any group element into products of generators the `S` and `T`. In particular this allows to check whether a given matrix indeed is a group element.

  It also allows one to calculate the automorphy factor of a modular form for the Hecke triangle group for arbitrary arguments.

  EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.hecke_triangle_groups import␣
↪HeckeTriangleGroup
sage: G = HeckeTriangleGroup(6)
sage: G.element_repr_method("basic")
sage: A = G.V(2)*G.V(3)^(-2)
sage: (L, sgn) = A.word_S_T()
sage: L
(S, T^(-2), S, T^(-1), S, T^(-1))
sage: sgn
-1
sage: sgn.parent()
Hecke triangle group for n = 6

sage: G(matrix([[-1, 1+G.lam()],[0, -1]]))
Traceback (most recent call last):
...
TypeError: The matrix is not an element of Hecke triangle group for n = 6, up to␣
↪equivalence it identifies two nonequivalent points.
sage: G(matrix([[-1, G.lam()],[0, -1]]))
-T^(-1)

sage: G.element_repr_method("basic")

sage: from sage.modular.modform_hecketriangle.space import ModularForms
sage: MF = ModularForms(G, k=4, ep=1)
sage: z = AlgebraicField()(1+i/2)
sage: MF.aut_factor(A, z)
37.62113890008...? + 12.18405525839...?*I
```

- **Representation of elements:** An element can be represented in several ways:
  - As a matrix over the base ring (default)
  - As a product of the generators `S` and `T`
  - As a product of basic blocks conjugated by some element

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.hecke_triangle_groups import␣
↪HeckeTriangleGroup
sage: G = HeckeTriangleGroup(n=5)
sage: el = G.S()*G.T(3)*G.S()*G.T(-2)

sage: G.element_repr_method("default")
sage: el
[        -1      2*lam]
[     3*lam -6*lam - 7]

sage: G.element_repr_method("basic")
sage: el
S*T^3*S*T^(-2)

sage: G.element_repr_method("block")
sage: el
-(S*T^3) * (V(4)^2*V(1)^3) * (S*T^3)^(-1)

sage: G.element_repr_method("conj")
sage: el
[-V(4)^2*V(1)^3]

sage: G.element_repr_method("default")
```

- **Group action on the (extended) upper half plane:** The group action of Hecke triangle groups on the (extended) upper half plane (by linear fractional transformations) is implemented. The implementation is not based on a specific upper half plane model but is defined formally (for arbitrary arguments) instead.

  It is possible to determine the group translate of an element in the classic (strict) fundamental domain for the group, together with the corresponding mapping group element.

  The corresponding action of the group on itself by conjugation is supported as well.

  The usual $slash$-operator for even integer weights is also available. It acts on rational functions (resp. polynomials). For modular forms an evaluation argument is required.

  EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.hecke_triangle_groups import␣
↪HeckeTriangleGroup
sage: G = HeckeTriangleGroup(n=7)
sage: G.element_repr_method("basic")

sage: G.S().acton(i + exp(-2))
-1/(e^(-2) + I)
sage: A = G.V(2)*G.V(3)^(-2)
sage: A
-S*T^(-2)*S*T^(-1)*S*T^(-1)
sage: A.acton(CC(i + exp(-2)))
0.344549645079... + 0.0163901095115...*I

sage: G.S().acton(A)
-T^(-2)*S*T^(-1)*S*T^(-1)*S

sage: z = AlgebraicField()(4 + 1/7*i)
sage: G.in_FD(z)
False
sage: (A, w) = G.get_FD(z)
```

```
sage: A
T^2*S*T^(-1)*S
sage: w
0.516937798396...? + 0.964078044600...?*I

sage: A.acton(w) == z
True
sage: G.in_FD(w)
True

sage: z = PolynomialRing(G.base_ring(), 'z').gen()
sage: rat = z^2 + 1/(z-G.lam())
sage: G.S().slash(rat)
(z^6 - lam*z^4 - z^3)/(-lam*z^4 - z^3)

sage: G.element_repr_method("default")
```

- **Basic properties of group elements:** The trace, sign (based on the trace), discriminant and elliptic/parabolic/hyperbolic type are available.

  Group elements can be displayed/represented in several ways:

  – As matrices over the base ring.

  – As a word in (powers of) the generators `S` and `T`.

  – As a word in (powers of) basic block matrices `V(j)` (resp. `U, S` in the elliptic case) together with the conjugation matrix that maps the element to this form (also see below).

  For the case `n=infinity` the last method is not properly implemented.

  EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.hecke_triangle_groups import␣
↪HeckeTriangleGroup
sage: G = HeckeTriangleGroup(n=7)
sage: A = -G.V(2)*G.V(3)^(-2)

sage: print(A.string_repr("default"))
[                lam          -lam^2 + 1]
[       2*lam^2 - 1 -2*lam^2 - lam + 2]
sage: print(A.string_repr("basic"))
S*T^(-2)*S*T^(-1)*S*T^(-1)
sage: print(A.string_repr("block"))
-(-S*T^(-1)*S) * (V(3)) * (-S*T^(-1)*S)^(-1)
sage: print(A.string_repr("conj"))
[-V(3)]
sage: A.trace()
-2*lam^2 + 2
sage: A.sign()
[-1  0]
[ 0 -1]
sage: A.discriminant()
4*lam^2 + 4*lam - 4
sage: A.is_elliptic()
False
sage: A.is_hyperbolic()
True
```

- **Fixed points:** Elliptic, parabolic or hyperbolic fixed points of group can be obtained. They are implemented as a (relative) quadratic extension (given by the square root of the discriminant) of the base ring. It is possible to query the correct embedding into a given field.

Note that for hyperbolic (and parabolic) fixed points there is a 1-1 correspondence with primitive hyperbolic/parabolic group elements (at least if `n < infinity`). The group action on fixed points resp. on matrices is compatible with this correpondence.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.hecke_triangle_groups import␣
↪HeckeTriangleGroup
sage: G = HeckeTriangleGroup(n=7)

sage: A = G.S()
sage: A.fixed_points()
(1/2*e, -1/2*e)
sage: A.fixed_points(embedded=True)
(I, -I)

sage: A = G.U()
sage: A.fixed_points()
(1/2*e + 1/2*lam, -1/2*e + 1/2*lam)
sage: A.fixed_points(embedded=True)
(0.9009688679024...? + 0.4338837391175...?*I, 0.9009688679024...? - 0.
↪4338837391175...?*I)

sage: A = -G.V(2)*G.V(3)^(-2)
sage: A.fixed_points()
((-3/7*lam^2 + 2/7*lam + 11/14)*e - 1/7*lam^2 + 3/7*lam + 3/7, (3/7*lam^2 - 2/
↪7*lam - 11/14)*e - 1/7*lam^2 + 3/7*lam + 3/7)
sage: A.fixed_points(embedded=True)
(0.3707208390178...?, 1.103231619181...?)

sage: el = A.fixed_points()[0]
sage: F = A.root_extension_field()
sage: F == el.parent()
True
sage: A.root_extension_embedding(CC)
Relative number field morphism:
  From: Number Field in e with defining polynomial x^2 - 4*lam^2 - 4*lam + 4 over␣
↪its base field
  To:   Complex Field with 53 bits of precision
  Defn: e |--> 4.02438434522465
        lam |--> 1.80193773580484

sage: G.V(2).acton(A).fixed_points()[0] == G.V(2).acton(el)
True
```

- **Lambda-continued fractions:** For parabolic or hyperbolic elements (resp. their corresponding fixed point) the (negative) lambda-continued fraction expansion is eventually periodic. The lambda-CF (i.e. the preperiod and period) is calculated exactly.

  In particular this allows to determine primitive and reduced generators of group elements and the corresponding primitive power of the element.

  The case `n=infinity` is not properly implemented.

  EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.hecke_triangle_groups import␣
↪HeckeTriangleGroup
sage: G = HeckeTriangleGroup(n=7)
sage: G.element_repr_method("block")

sage: G.V(6).continued_fraction()
((1,), (1, 1, 1, 1, 2))
sage: (-G.V(2)).continued_fraction()
((1,), (2,))

sage: A = -(G.V(2)*G.V(3)^(-2))^2
sage: A.is_primitive()
False
sage: A.primitive_power()
2
sage: A.is_reduced()
False
sage: A.continued_fraction()
((1, 1, 1, 1), (1, 2))

sage: B = A.primitive_part()
sage: B
(-S*T^(-1)*S) * (V(3)) * (-S*T^(-1)*S)^(-1)
sage: B.is_primitive()
True
sage: B.is_reduced()
False
sage: B.continued_fraction()
((1, 1, 1, 1), (1, 2))
sage: A == A.sign() * B^A.primitive_power()
True

sage: B = A.reduce()
sage: B
(T*S*T) * (V(3)) * (T*S*T)^(-1)
sage: B.is_primitive()
True
sage: B.is_reduced()
True
sage: B.continued_fraction()
((), (1, 2))

sage: G.element_repr_method("default")
```

- **Reduced and simple elements, Hecke-symmetric elements:** For primitive conjugacy classes of hyperbolic elements the cycle of reduced elements can be obtain as well as all simple elements. It is also possible to determine whether a class is Hecke-symmetric.

  The case `n=infinity` is not properly implemented.

  EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.hecke_triangle_groups import␣
↪HeckeTriangleGroup
sage: G = HeckeTriangleGroup(n=5)

sage: el = G.V(1)^2*G.V(2)*G.V(4)
sage: R = el.reduced_elements()
```

```
sage: [v.continued_fraction() for v in R]
[((), (2, 1, 1, 4)), ((), (1, 1, 4, 2)), ((), (1, 4, 2, 1)), ((), (4, 2, 1, 1))]

sage: el = G.V(1)^2*G.V(2)*G.V(4)
sage: R = el.simple_elements()
sage: [v.is_simple() for v in R]
[True, True, True, True]
sage: (fp1, fp2) = R[2].fixed_points(embedded=True)
sage: fp2 < 0 < fp1
True

sage: el = G.V(2)
sage: el.is_hecke_symmetric()
False
sage: (el.simple_fixed_point_set(), el.inverse().simple_fixed_point_set())
({1/2*e, (-1/2*lam + 1/2)*e}, {(1/2*lam - 1/2)*e, -1/2*e})
sage: el = G.V(2)*G.V(3)
sage: el.is_hecke_symmetric()
True
sage: el.simple_fixed_point_set() == el.inverse().simple_fixed_point_set()
True
```

- **Rational period functions:** For each primitive (hyperbolic) conjugacy classes and each even weight `k` we can associate a corresponding rational period function. I.e. a rational function `q` of weight `k` which satisfies: `q | S == 0` and `q + q|U + ... + q|U^(n-1) == 0`, where `S`, `U` are the corresponding group elements and `|` is the usual $slash-operator$ of weight `k`.

  The set of all rational period function is expected to be generated by such functions.

  EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.hecke_triangle_groups import␣
↪HeckeTriangleGroup
sage: G = HeckeTriangleGroup(n=5)
sage: S = G.S()
sage: U = G.U()

sage: def is_rpf(f, k=None):
....:     if not f + S.slash(f, k=k) == 0:
....:         return False
....:     if not sum([(U^m).slash(f, k=k) for m in range(G.n())]) == 0:
....:         return False
....:     return True

sage: z = PolynomialRing(G.base_ring(), 'z').gen()
sage: uniq([ is_rpf(1 - z^(-k), k=k) for k in range(-6, 6, 2)])     # long time
[True]
sage: [is_rpf(1/z, k=k) for k in range(-6, 6, 2)]
[False, False, False, False, True, False]

sage: el = G.V(2)
sage: el.is_hecke_symmetric()
False
sage: rpf = el.rational_period_function(-4)
sage: is_rpf(rpf)
True
sage: rpf
-lam*z^4 + lam
```

```
sage: rpf = el.rational_period_function(-2)
sage: is_rpf(rpf)
True
sage: rpf
(lam + 1)*z^2 - lam - 1
sage: el.rational_period_function(0) == 0
True
sage: rpf = el.rational_period_function(2)
sage: is_rpf(rpf)
True
sage: rpf
((lam + 1)*z^2 - lam - 1)/(lam*z^4 + (-lam - 2)*z^2 + lam)

sage: el = G.V(2)*G.V(3)
sage: el.is_hecke_symmetric()
True
sage: el.rational_period_function(-4) == 0
True
sage: rpf = el.rational_period_function(-2)
sage: rpf
(8*lam + 4)*z^2 - 8*lam - 4
sage: rpf = el.rational_period_function(2)
sage: is_rpf(rpf)
True
sage: rpf.denominator()
(144*lam + 89)*z^8 + (-618*lam - 382)*z^6 + (951*lam + 588)*z^4 + (-618*lam -␣
↪382)*z^2 + 144*lam + 89
sage: el.rational_period_function(4) == 0
True

sage: G = HeckeTriangleGroup(n=4)
sage: G.rational_period_functions(k=4, D=12)
[(z^4 - 1)/z^4]
sage: G.rational_period_functions(k=2, D=14)
[(z^2 - 1)/z^2, 1/z, (24*z^6 - 120*z^4 + 120*z^2 - 24)/(9*z^8 - 80*z^6 + 146*z^4 -
↪ 80*z^2 + 9), (24*z^6 - 120*z^4 + 120*z^2 - 24)/(9*z^8 - 80*z^6 + 146*z^4 -␣
↪80*z^2 + 9)]
```

- **Block decomposition of elements:** For each group element a very specific conjugacy representative can be obtained. For hyperbolic and parabolic elements the representative is a product `V(j)` -matrices. They all have non-negative trace and the number of factors is called the block length of the element (which is implemented).

  Note: For this decomposition special care is given to the sign (of the trace) of the matrices.

  The case `n=infinity` for everything above is not properly implemented.

  EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.hecke_triangle_groups import␣
↪HeckeTriangleGroup
sage: G = HeckeTriangleGroup(n=7)
sage: G.element_repr_method("block")

sage: A = -G.V(2)*G.V(6)^3*G.V(3)
sage: A
-(T*S*T) * (V(6)^3*V(3)*V(2)) * (T*S*T)^(-1)
sage: A.sign()
-1
sage: (L, R, sgn) = A.block_decomposition()
```

```
sage: L
((-S*T^(-1)*S) * (V(6)^3) * (-S*T^(-1)*S)^(-1), (T*S*T*S*T) * (V(3)) *␣
↪(T*S*T*S*T)^(-1), (T*S*T) * (V(2)) * (T*S*T)^(-1))
sage: prod(L).sign()
1
sage: A == sgn * (R.acton(prod(L)))
True
sage: t = A.block_length()
sage: t
5
sage: AA(A.discriminant()) >= AA(t^2 * G.lam() - 4)
True
```

- **Class number and class representatives**: The block length provides a lower bound for the discriminant. This allows to enlist all (representatives of) matrices of (or up to) a given discriminant.

  Using the 1-1 correspondence with hyperbolic fixed points (and certain hyperbolic binary quadratic forms) this makes it possible to calculate the corresponding class number (number of conjugacy classes for a given discriminant).

  It also allows to list all occurring discriminants up to some bound. Or to enlist all reduced/simple elements resp. their corresponding hyperbolic fixed points for the given discriminant.

  Warning: The currently used algorithm is very slow!

  EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.hecke_triangle_groups import␣
↪HeckeTriangleGroup
sage: G = HeckeTriangleGroup(n=4)
sage: G.element_repr_method("basic")
sage: G.is_discriminant(68)
True
sage: G.class_number(14)
2
sage: G.list_discriminants(D=68)
[4, 12, 14, 28, 32, 46, 60, 68]
sage: G.list_discriminants(D=0, hyperbolic=False, primitive=False)
[-4, -2, 0]
sage: G.class_number(68)
4
sage: G.class_representatives(68)
[S*T^(-2)*S*T^(-1)*S*T, -S*T^(-1)*S*T^2*S*T, S*T^(-5)*S*T^(-1)*S, T*S*T^5]
sage: R = G.reduced_elements(68)
sage: uniq([v.is_reduced() for v in R])     # long time
[True]
sage: R = G.simple_elements(68)
sage: uniq([v.is_simple() for v in R])     # long time
[True]
sage: G.element_repr_method("default")

sage: G = HeckeTriangleGroup(n=5)
sage: G.element_repr_method("basic")
sage: G.list_discriminants(9*G.lam() + 5)
[4*lam, 7*lam + 6, 9*lam + 5]
sage: G.list_discriminants(D=0, hyperbolic=False, primitive=False)
[-4, -lam - 2, lam - 3, 0]
sage: G.class_number(9*G.lam() + 5)
2
```

```
sage: G.class_representatives(9*G.lam() + 5)
[S*T^(-2)*S*T^(-1)*S, T*S*T^2]
sage: R = G.reduced_elements(9*G.lam() + 5)
sage: uniq([v.is_reduced() for v in R])     # long time
[True]
sage: R = G.simple_elements(7*G.lam() + 6)
sage: for v in R: print(v.string_repr("default"))
[lam + 2      lam]
[    lam        1]
[      1      lam]
[    lam lam + 2]
sage: G.element_repr_method("default")
```

1.2 Modular forms ring and spaces for Hecke triangle groups:

- **Analytic type:** The analytic type of forms, including the behavior at infinity:
  - Meromorphic (and meromorphic at infinity)
  - Weakly holomorphic (holomorphic and meromorphic at infinity)
  - Holomorphic (and holomorphic at infinity)
  - Cuspidal (holomorphic and zero at infinity)

  Additionally the type specifies whether the form is modular or only quasi modular.

  EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.analytic_type import AnalyticType
sage: AnalyticType()(["quasi", "cusp"])
quasi cuspidal
```

- **Modular form (for Hecke triangle groups):** A function of some analytic type which transforms like a modular form for the given group, weight `k` and multiplier `epsilon`:
  - `f(z+lambda) = f(lambda)`
  - `f(-1/z) = epsilon * (z/i)^k * f(z)`

  The multiplier is either `1` or `-1`. The weight is a rational number of the form `4*(n*l+l')/(n-2) + (1-epsilon)*n/(n-2)`. If `n` is odd, then the multiplier is unique and given by `(-1)^(k*(n-2)/2)`. The space of modular forms for a given group, weight and multiplier forms a module over the base ring. It is finite dimensional if the analytic type is `holomorphic`.

  Modular forms can be constructed in several ways:
  - Using some already available construction function for modular forms (those function are available for all spaces/rings and in general do not return elements of the same parent)
  - Specifying the form as a rational function in the basic generators (see below)
  - For weakly holomorphic modular forms it is possible to exactly determine the form by specifying (sufficiently many) initial coefficients of its Fourier expansion.
  - There is even hope (no garantuee) to determine a (exact) form from the initial numerical coefficients (see below).
  - By specifying the coefficients with respect to a basis of the space (if the corresponding space supports coordinate vectors)

  - Arithmetic combination of forms or differential operators applied to forms

The implementation is based on the implementation of the graded ring (see below). All calculations are exact (no precision argument is required). The analytic type of forms is checked during construction. The analytic type of parent spaces after arithmetic/differential operations with elements is changed (extended/reduced) accordingly.

In particular it is possible to multiply arbitrary modular forms (and end up with an element of a modular forms space). If two forms of different weight/multiplier are added then an element of the corresponding modular forms ring is returned instead.

Elements of modular forms spaces are represented by their Fourier expansion.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.space import CuspForms,␣
↪ModularForms, MeromorphicModularForms
sage: MeromorphicModularForms(n=4, k=8, ep=1)
MeromorphicModularForms(n=4, k=8, ep=1) over Integer Ring
sage: CF = CuspForms(n=7, k=12, ep=1)
sage: CF
CuspForms(n=7, k=12, ep=1) over Integer Ring

sage: MF = ModularForms(k=12, ep=1)
sage: (x,y,z,d) = MF.pol_ring().gens()

Using existing functions:
sage: CF.Delta()
q + 17/(56*d)*q^2 + 88887/(2458624*d^2)*q^3 + 941331/(481890304*d^3)*q^4 + O(q^5)

Using rational function in the basic generators:
sage: MF(x^3)
1 + 720*q + 179280*q^2 + 16954560*q^3 + 396974160*q^4 + O(q^5)

Using Fourier expansions:
sage: qexp = CF.Delta().q_expansion(prec=2)
sage: qexp
q + O(q^2)
sage: qexp.parent()
Power Series Ring in q over Fraction Field of Univariate Polynomial Ring in d␣
↪over Integer Ring
sage: MF(qexp)
q - 24*q^2 + 252*q^3 - 1472*q^4 + O(q^5)

Using coordinate vectors:
sage: MF([0,1]) == MF.f_inf()
True

Using arithmetic expressions:
sage: d = CF.get_d()
sage: CF.f_rho()^7 / (d*CF.f_rho()^7 - d*CF.f_i()^2) == CF.j_inv()
True
sage: MF.E4().serre_derivative() == -1/3 * MF.E6()
True
```

- **Hauptmodul:** The `j-function` for Hecke triangle groups is given by the unique Riemann map from the hyperbolic triangle with vertices at `rho`, `i` and `infinity` to the upper half plane, normalized such that its Fourier coefficients are real and such that the first nontrivial Fourier coefficient is 1. The function extends to a completely invariant weakly holomorphic function from the upper half plane to the complex numbers. Another used normalization (in capital letters) is `J(i)=1`. The coefficients of `j` are rational numbers up to a power of d=1/j(i) which is only rational in the arithmetic cases n=3,4,6,infinity.

All Fourier coefficients of modular forms are based on the coefficients of j. The coefficients of j are calculated by inverting the Fourier series of its inverse (the series inversion is also by far the most expensive operation of all).

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.graded_ring import␣
↪WeakModularFormsRing
sage: from sage.modular.modform_hecketriangle.space import WeakModularForms
sage: WeakModularForms(n=3, k=0, ep=1).j_inv()
q^-1 + 744 + 196884*q + 21493760*q^2 + 864299970*q^3 + 20245856256*q^4 + O(q^5)
sage: WeakModularFormsRing(n=7).j_inv()
f_rho^7/(f_rho^7*d - f_i^2*d)
sage: WeakModularFormsRing(n=7, red_hom=True).j_inv()
q^-1 + 151/(392*d) + 165229/(2458624*d^2)*q + 107365/(15059072*d^3)*q^2 +␣
↪25493858865/(48358655787008*d^4)*q^3 + 2771867459/(92561489592320*d^5)*q^4 +␣
↪O(q^5)
```

- **Basic generators:** There exist unique modular forms f_rho, f_i and f_inf such that each has a simple zero at rho=exp(pi/n), i and infinity resp. and no other zeros. The forms are normalized such that their first Fourier coefficient is 1. They have the weight and multiplier (4/(n-2),1), (2*n/(n-2),-1), (4*n/(n-2),1) resp. and can be defined in terms of the Hauptmodul j.

  EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.graded_ring import ModularFormsRing
sage: ModularFormsRing(n=5, red_hom=True).f_rho()
1 + 7/(100*d)*q + 21/(160000*d^2)*q^2 + 1043/(192000000*d^3)*q^3 + 45479/
↪(1228800000000*d^4)*q^4 + O(q^5)
sage: ModularFormsRing(n=5, red_hom=True).f_i()
1 - 13/(40*d)*q - 351/(64000*d^2)*q^2 - 13819/(76800000*d^3)*q^3 - 1163669/
↪(491520000000*d^4)*q^4 + O(q^5)
sage: ModularFormsRing(n=5, red_hom=True).f_inf()
q - 9/(200*d)*q^2 + 279/(640000*d^2)*q^3 + 961/(192000000*d^3)*q^4 + O(q^5)
sage: ModularFormsRing(n=5).f_inf()
f_rho^5*d - f_i^2*d
```

- **Eisenstein series and Delta:** The Eisenstein series of weight 2, 4 and 6 exist for all n and are all implemented . Note that except for n=3 the series E4 and E6 do not coincide with f_rho and f_i.

  Similarly there always exists a (generalization of) Delta. Except for n=3 it also does not coincide with f_inf.

  In general Eisenstein series of all even weights exist for all n. In the non-arithmetic cases they are however very hard to determine (it's an open problem(?) and consequently not yet implemented, except for trivial one-dimensional cases).

  The Eisenstein series in the arithmetic cases n = 3,4,6 are fully implemented though. Note that this requires a lot more work/effort for k != 2,4,6 resp. for multidimensional spaces.

  The case n=infinity is a special case (since there are two cusps) and is not implemented yet.

  EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.graded_ring import ModularFormsRing
sage: from sage.modular.modform_hecketriangle.space import ModularForms
sage: ModularFormsRing(n=5).E4()
f_rho^3
sage: ModularFormsRing(n=5).E6()
```

```
f_rho^2*f_i
sage: ModularFormsRing(n=5).Delta()
f_rho^9*d - f_rho^4*f_i^2*d
sage: ModularFormsRing(n=5).Delta() == ModularFormsRing(n=5).f_
↪inf()*ModularFormsRing(n=5).f_rho()^4
True

The basic generators in some arithmetic cases:
sage: ModularForms(n=3, k=6).E6()
1 - 504*q - 16632*q^2 - 122976*q^3 - 532728*q^4 + O(q^5)
sage: ModularForms(n=4, k=6).E6()
1 - 56*q - 2296*q^2 - 13664*q^3 - 73976*q^4 + O(q^5)
sage: ModularForms(n=infinity, k=4).E4()
1 + 16*q + 112*q^2 + 448*q^3 + 1136*q^4 + O(q^5)

General Eisenstein series in some arithmetic cases:
sage: ModularFormsRing(n=4).EisensteinSeries(k=8)
(-25*f_rho^4 - 9*f_i^2)/(-34)
sage: ModularForms(n=3, k=12).EisensteinSeries()
1 + 65520/691*q + 134250480/691*q^2 + 11606736960/691*q^3 + 274945048560/691*q^4␣
↪+ O(q^5)
sage: ModularForms(n=6, k=12).EisensteinSeries()
1 + 6552/50443*q + 13425048/50443*q^2 + 1165450104/50443*q^3 + 27494504856/
↪50443*q^4 + O(q^5)
sage: ModularForms(n=4, k=22, ep=-1).EisensteinSeries()
1 - 184/53057489*q - 386252984/53057489*q^2 - 1924704989536/53057489*q^3 -␣
↪810031218278584/53057489*q^4 + O(q^5)
```

- **Generator for ‘‘k=0‘‘, ‘‘ep=-1‘‘:** If `n` is even then the space of weakly holomorphic modular forms of weight `0` and multiplier `-1` is not empty and generated by one element, denoted by `g_inv` .

  EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.space import WeakModularForms
sage: WeakModularForms(n=4, k=0, ep=-1).g_inv()
q^-1 - 24 - 3820*q - 100352*q^2 - 1217598*q^3 - 10797056*q^4 + O(q^5)
sage: WeakModularFormsRing(n=8).g_inv()
f_rho^4*f_i/(f_rho^8*d - f_i^2*d)
```

- **Quasi modular form (for Hecke triangle groups):** `E2` no longer transforms like a modular form but like a quasi modular form. More generally quasi modular forms are given in terms of modular forms and powers of `E2` . E.g. a holomorphic quasi modular form is a sum of holomorphic modular forms multiplied with a power of `E2` such that the weights and multipliers match up. The space of quasi modular forms for a given group, weight and multiplier forms a module over the base ring. It is finite dimensional if the analytic type is `holomorphic` .

  The implementation and construction are analogous to modular forms (see above). In particular construction of quasi weakly holomorphic forms by their initial Laurent coefficients is supported as well!

  EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.graded_ring import ModularFormsRing
sage: from sage.modular.modform_hecketriangle.space import QuasiCuspForms,␣
↪QuasiModularForms, QuasiWeakModularForms
sage: QuasiCuspForms(n=7, k=12, ep=1)
QuasiCuspForms(n=7, k=12, ep=1) over Integer Ring
sage: QuasiModularForms(n=4, k=8, ep=-1)
QuasiModularForms(n=4, k=8, ep=-1) over Integer Ring
```

```
sage: QuasiModularForms(n=4, k=2, ep=-1).E2()
1 - 8*q - 40*q^2 - 32*q^3 - 104*q^4 + O(q^5)

A quasi weak form can be constructed by using its initial Laurent expansion:
sage: QF = QuasiWeakModularForms(n=8, k=10/3, ep=-1)
sage: qexp = (QF.quasi_part_gens(min_exp=-1)[4]).q_expansion(prec=5)
sage: qexp
q^-1 - 19/(64*d) - 7497/(262144*d^2)*q + 15889/(8388608*d^3)*q^2 + 543834047/
↪(1649267441664*d^4)*q^3 + 711869853/(43980465111040*d^5)*q^4 + O(q^5)
sage: qexp.parent()
Laurent Series Ring in q over Fraction Field of Univariate Polynomial Ring in d␣
↪over Integer Ring
sage: QF(qexp).as_ring_element()
f_rho^3*f_i*E2^2/(f_rho^8*d - f_i^2*d)
sage: QF(qexp).reduced_parent()
QuasiWeakModularForms(n=8, k=10/3, ep=-1) over Integer Ring

Derivatives of (quasi weak) modular forms are again quasi (weak) modular forms:
sage: CF.f_inf().derivative() == CF.f_inf()*CF.E2()
True
```

- **Ring of (quasi) modular forms:** The ring of (quasi) modular forms for a given analytic type and Hecke triangle group. In fact it is a graded algebra over the base ring where the grading is over `1/(n-2)*Z x Z/(2Z)` corresponding to the weight and multiplier. A ring element is thus a finite linear combination of (quasi) modular forms of (possibly) varying weights and multipliers.

  Each ring element is represented as a rational function in the generators `f_rho` , `f_i` and `E2` . The representations and arithmetic operations are exact (no precision argument is required).

  Elements of the ring are represented by the rational function in the generators.

  If the parameter `red_hom` is set to `True` (default: `False`) then operations with homogeneous elements try to return an element of the corresponding vector space (if the element is homogeneous) instead of the forms ring. It is also easier to use the forms ring with `red_hom=True` to construct known forms (since then it is not required to specify the weight and multiplier).

  EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.graded_ring import␣
↪QuasiModularFormsRing, ModularFormsRing
sage: QuasiModularFormsRing(n=5, red_hom=True)
QuasiModularFormsRing(n=5) over Integer Ring
sage: ModularFormsRing()
ModularFormsRing(n=3) over Integer Ring
sage: (x,y,z,d) = ModularFormsRing().pol_ring().gens()

sage: ModularFormsRing()(x+y)
f_rho + f_i

sage: QuasiModularFormsRing(n=5, red_hom=True)(x^5-y^2).reduce()
1/d*q - 9/(200*d^2)*q^2 + 279/(640000*d^3)*q^3 + 961/(192000000*d^4)*q^4 + O(q^5)
```

- **Construction of modular forms spaces and rings:** There are functorial constructions behind all forms spaces and rings which assure that arithmetic operations between those spaces and rings work and fit into the coercion framework. In particular ring elements are interpreted as constant modular forms in this context and base extensions are done if necessary.
- **Fourier expansion of (quasi) modular forms (for Hecke triangle groups):** Each (quasi) modular form (in fact

each ring element) possesses a Fourier expansion of the form `sum_{n>=n_0} a_n q^n`, where `n_0` is an integer, `q=exp(2*pi*i*z/lambda)` and the coefficients `a_n` are rational numbers (or more generally an extension of rational numbers) up to a power of `d`, where `d` is the (possibly) transcendental parameter described above. I.e. the coefficient ring is given by `Frac(R)(d)`.

The coefficients are calculated exactly in terms of the (formal) parameter `d`. The expansion is calculated exactly up to the specified precision. It is also possible to get a Fourier expansion where `d` is evaluated to its numerical approximation.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.graded_ring import␣
↪ModularFormsRing, QuasiModularFormsRing
sage: ModularFormsRing(n=4).j_inv().q_expansion(prec=3)
q^-1 + 13/(32*d) + 1093/(16384*d^2)*q + 47/(8192*d^3)*q^2 + O(q^3)
sage: QuasiModularFormsRing(n=5).E2().q_expansion(prec=3)
1 - 9/(200*d)*q - 369/(320000*d^2)*q^2 + O(q^3)
sage: QuasiModularFormsRing(n=5).E2().q_expansion_fixed_d(prec=3)
1.000000000000... - 6.380956565426...*q - 23.18584547617...*q^2 + O(q^3)
```

- **Evaluation of forms:** (Quasi) modular forms (and also ring elements) can be viewed as functions from the upper half plane and can be numerically evaluated by using the Fourier expansion.

  The evaluation uses the (quasi) modularity properties (if possible) for a faster and more precise evaluation. The precision of the result depends both on the numerical precision and on the default precision used for the Fourier expansion.

  EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.graded_ring import ModularFormsRing
sage: f_i = ModularFormsRing(n=4).f_i()
sage: f_i(i)
0
sage: f_i(infinity)
1
sage: f_i(1/7 + 0.01*i)
32189.02016723... + 21226.62951394...*I
```

- **L-functions of forms:** Using the (pari based) function `Dokchitser` L-functions of non-constant holomorphic modular forms are supported for all values of `n`.

  Note: For non-arithmetic groups this involves an irrational conductor. The conductor for the arithmetic groups `n = 3,4,6,infinity` is `1,2,3,4` respectively.

  EXAMPLES:

```
sage: from sage.modular.modform.eis_series import eisenstein_series_lseries
sage: from sage.modular.modform_hecketriangle.space import ModularForms
sage: f = ModularForms(n=3, k=4).E4()/240
sage: L = f.lseries()
sage: L.conductor
1
sage: L.check_functional_equation() < 2^(-50)
True
sage: L(1)
-0.0304484570583...
sage: abs(L(1) - eisenstein_series_lseries(4)(1)) < 2^(-53)
True
sage: L.taylor_series(1, 3)
-0.0304484570583... - 0.0504570844798...*z - 0.0350657360354...*z^2 + O(z^3)
```

```
sage: coeffs = f.q_expansion_vector(min_exp=0, max_exp=20, fix_d=True)
sage: abs(L(10) - sum([coeffs[k]*k^(-10) for k in range(1,len(coeffs))]).n(53)) <␣
↪10^(-7)
True

sage: L = ModularForms(n=6, k=6, ep=-1).E6().lseries(num_prec=200)
sage: L.conductor
3
sage: L.check_functional_equation() < 2^(-180)
True
sage: L.eps
-1
sage: abs(L(3)) < 2^(-180)
True

sage: L = ModularForms(n=17, k=12).Delta().lseries()
sage: L.conductor
3.86494445880...
sage: L.check_functional_equation() < 2^(-50)
True
sage: L.taylor_series(6, 3)
2.15697985314... - 1.17385918996...*z + 0.605865993050...*z^2 + O(z^3)

sage: L = ModularForms(n=infinity, k=2, ep=-1).f_i().lseries()
sage: L.conductor
4
sage: L.check_functional_equation() < 2^(-50)
True
sage: L.taylor_series(1, 3)
0.000000000000... + 5.76543616701...*z + 9.92776715593...*z^2 + O(z^3)
```

- **(Serre) derivatives:** Derivatives and Serre derivatives of forms can be calculated. The analytic type is extended accordingly.

  EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.graded_ring import ModularFormsRing
sage: from sage.modular.modform_hecketriangle.space import QuasiModularForms
sage: f_inf = ModularFormsRing(n=4, red_hom=True).f_inf()
sage: f_inf.derivative()/f_inf == QuasiModularForms(n=4, k=2, ep=-1).E2()
True
sage: ModularFormsRing().E4().serre_derivative() == -1/3 * ModularFormsRing().E6()
True
```

- **Basis for weakly holomorphic modular forms and Faber polynomials:** (Natural) generators of weakly holomorphic modular forms can be obtained using the corresponding generalized Faber polynomials.

  EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.space import WeakModularForms,␣
↪CuspForms
sage: MF = WeakModularForms(n=5, k=62/3, ep=-1)
sage: MF.disp_prec(MF._l1+2)

sage: MF.F_basis(2)
q^2 - 41/(200*d)*q^3 + O(q^4)
sage: MF.F_basis(1)
q - 13071/(640000*d^2)*q^3 + O(q^4)
```

```
sage: MF.F_basis(-0)
1 - 277043/(192000000*d^3)*q^3 + O(q^4)
sage: MF.F_basis(-2)
q^-2 - 162727620113/(40960000000000000*d^5)*q^3 + O(q^4)
```

- **Basis for quasi weakly holomorphic modular forms:** (Natural) generators of quasi weakly holomorphic modular forms can also be obtained. In most cases it is even possible to find a basis consisting of elements with only one non-trivial Laurent coefficient (up to some coefficient).

  EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.space import QuasiWeakModularForms
sage: QF = QuasiWeakModularForms(n=8, k=10/3, ep=-1)
sage: QF.default_prec(1)
sage: QF.quasi_part_gens(min_exp=-1)
[q^-1 + O(q),
 1 + O(q),
 q^-1 - 9/(128*d) + O(q),
 1 + O(q),
 q^-1 - 19/(64*d) + O(q),
 q^-1 + 1/(64*d) + O(q)]
sage: QF.default_prec(QF.required_laurent_prec(min_exp=-1))
sage: QF.q_basis(min_exp=-1)     # long time
[q^-1 + O(q^5),
 1 + O(q^5),
 q + O(q^5),
 q^2 + O(q^5),
 q^3 + O(q^5),
 q^4 + O(q^5)]
```

- **Dimension and basis for holomorphic or cuspidal (quasi) modular forms:** For finite dimensional spaces the dimension and a basis can be obtained.

  EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.space import QuasiModularForms
sage: MF = QuasiModularForms(n=5, k=6, ep=-1)
sage: MF.dimension()
3
sage: MF.default_prec(2)
sage: MF.gens()
[1 - 37/(200*d)*q + O(q^2),
 1 + 33/(200*d)*q + O(q^2),
 1 - 27/(200*d)*q + O(q^2)]
```

- **Coordinate vectors for (quasi) holomorphic modular forms and (quasi) cusp forms:** For (quasi) holomorphic modular forms and (quasi) cusp forms it is possible to determine the coordinate vectors of elements with respect to the basis.

  EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.space import ModularForms
sage: ModularForms(n=7, k=12, ep=1).dimension()
3
sage: ModularForms(n=7, k=12, ep=1).Delta().coordinate_vector()
(0, 1, 17/(56*d))

sage: from sage.modular.modform_hecketriangle.space import QuasiCuspForms
sage: MF = QuasiCuspForms(n=7, k=20, ep=1)
```

```
sage: MF.dimension()
13
sage: el = MF(MF.Delta()*MF.E2()^4 + MF.Delta()*MF.E2()*MF.E6())
sage: el.coordinate_vector()     # long time
(0, 0, 0, 1, 29/(196*d), 0, 0, 0, 0, 1, 17/(56*d), 0, 0)
```

- **Subspaces:** It is possible to construct subspaces of (quasi) holomorphic modular forms or (quasi) cusp forms spaces with respect to a specified basis of the corresponding ambient space. The subspaces also support coordinate vectors with respect to its basis.

  EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.space import ModularForms
sage: MF = ModularForms(n=7, k=12, ep=1)
sage: subspace = MF.subspace([MF.E4()^3, MF.Delta()])
sage: subspace
Subspace of dimension 2 of ModularForms(n=7, k=12, ep=1) over Integer Ring
sage: el = subspace(MF.E6()^2)
sage: el.coordinate_vector()
(1, -61/(196*d))
sage: el.ambient_coordinate_vector()
(1, -61/(196*d), -51187/(614656*d^2))

sage: from sage.modular.modform_hecketriangle.space import QuasiCuspForms
sage: MF = QuasiCuspForms(n=7, k=20, ep=1)
sage: subspace = MF.subspace([MF.Delta()*MF.E2()^2*MF.E4(), MF.Delta()*MF.E2()^
↪4])     # long time
sage: subspace     # long time
Subspace of dimension 2 of QuasiCuspForms(n=7, k=20, ep=1) over Integer Ring
sage: el = subspace(MF.Delta()*MF.E2()^4)     # long time
sage: el.coordinate_vector()     # long time
(0, 1)
sage: el.ambient_coordinate_vector()     # long time
(0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 17/(56*d), 0, 0)
```

- **Theta subgroup:** The Hecke triangle group corresponding to `n=infinity` is also completely supported. In particular the (special) behavior around the cusp `-1` is considered and can be specified.

  EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.graded_ring import␣
↪QuasiMeromorphicModularFormsRing
sage: MR = QuasiMeromorphicModularFormsRing(n=infinity, red_hom=True)
sage: MR
QuasiMeromorphicModularFormsRing(n=+Infinity) over Integer Ring
sage: j_inv = MR.j_inv().full_reduce()
sage: f_i = MR.f_i().full_reduce()
sage: E4 = MR.E4().full_reduce()
sage: E2 = MR.E2().full_reduce()

sage: j_inv
q^-1 + 24 + 276*q + 2048*q^2 + 11202*q^3 + 49152*q^4 + O(q^5)
sage: MR.f_rho() == MR(1)
True
sage: E4
1 + 16*q + 112*q^2 + 448*q^3 + 1136*q^4 + O(q^5)
sage: f_i
1 - 24*q + 24*q^2 - 96*q^3 + 24*q^4 + O(q^5)
```

```
sage: E2
1 - 8*q - 8*q^2 - 32*q^3 - 40*q^4 + O(q^5)
sage: E4.derivative() == E4 * (E2 - f_i)
True
sage: f_i.serre_derivative() == -1/2 * E4
True
sage: MF = f_i.serre_derivative().parent()
sage: MF
ModularForms(n=+Infinity, k=4, ep=1) over Integer Ring
sage: MF.dimension()
2
sage: MF.gens()
[1 + 240*q^2 + 2160*q^4 + O(q^5), q - 8*q^2 + 28*q^3 - 64*q^4 + O(q^5)]
sage: E4(i)
1.941017189...
sage: E4.order_at(-1)
1

sage: MF = (E2/E4).reduced_parent()
sage: MF.quasi_part_gens(order_1=-1)
[1 - 40*q + 552*q^2 - 4896*q^3 + 33320*q^4 + O(q^5),
 1 - 24*q + 264*q^2 - 2016*q^3 + 12264*q^4 + O(q^5)]
sage: prec = MF.required_laurent_prec(order_1=-1)
sage: qexp = (E2/E4).q_expansion(prec=prec)
sage: qexp
1 - 3/(8*d)*q + O(q^2)
sage: MF.construct_quasi_form(qexp, order_1=-1) == E2/E4
True
sage: MF.disp_prec(6)
sage: MF.q_basis(m=-1, order_1=-1, min_exp=-1)
q^-1 - 203528/7*q^5 + O(q^6)
```

Elements with respect to the full group are automatically coerced to elements of the Theta subgroup if necessary:

```
sage: el = QuasiMeromorphicModularFormsRing(n=3).Delta().full_reduce() + E2
sage: el
(E4*f_i^4 - 2*E4^2*f_i^2 + E4^3 + 4096*E2)/4096
sage: el.parent()
QuasiModularFormsRing(n=+Infinity) over Integer Ring
```

- **Determine exact coefficients from numerical ones:** There is some experimental support for replacing numerical coefficients with corresponding exact coefficients. There is however NO guarantee that the procedure will work (and most probably there are cases where it won't).

  EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.space import WeakModularForms,␣
↪QuasiCuspForms
sage: WF = WeakModularForms(n=14)
sage: qexp = WF.J_inv().q_expansion_fixed_d(d_num_prec=1000)
sage: qexp.parent()
Laurent Series Ring in q over Real Field with 1000 bits of precision
sage: qexp_int = WF.rationalize_series(qexp)
doctest:...: UserWarning: Using an experimental rationalization of coefficients,␣
↪please check the result for correctness!
sage: qexp_int.parent()
Laurent Series Ring in q over Fraction Field of Univariate Polynomial Ring in d␣
↪over Integer Ring
```

```
sage: qexp_int == WF.J_inv().q_expansion()
True
sage: WF(qexp_int) == WF.J_inv()
True

sage: QF = QuasiCuspForms(n=8, k=22/3, ep=-1)
sage: el = QF(QF.f_inf()*QF.E2())
sage: qexp = el.q_expansion_fixed_d(d_num_prec=1000)
sage: qexp_int = QF.rationalize_series(qexp)
sage: qexp_int == el.q_expansion()
True
sage: QF(qexp_int) == el
True
```

1.3 Future ideas:

- Complete support for the case `n=infinity` (e.g. lambda-CF)
- Properly implemented lambda-CF
- Binary quadratic forms for Hecke triangle groups
- Cycle integrals
- Maybe: Proper spaces (with coordinates) for (quasi) weakly holomorphic forms with bounds on the initial Fourier exponent
- Support for general triangle groups (hard)
- Support for “congruence” subgroups (hard)

CHAPTER

TWO

GRADED RINGS OF MODULAR FORMS FOR HECKE TRIANGLE GROUPS

AUTHORS:

- Jonas Jermann (2013): initial version

class `sage.modular.modform_hecketriangle.abstract_ring.`**`FormsRing_abstract`**`(`*group*, *base_ring*, *red_hom*, *n*`)`

Bases: `sage.structure.parent.Parent`

Abstract (Hecke) forms ring.

This should never be called directly. Instead one should instantiate one of the derived classes of this class.

`AnalyticType`

alias of *`AnalyticType`*

`Delta``()`

Return an analog of the Delta-function.

It lies in the graded ring of `self`. In case `has_reduce_hom` is `True` it is given as an element of the corresponding space of homogeneous elements.

It is a cusp form of weight 12 and is equal to `d*(E4^3 -E6^2)` or (in terms of the generators) `d*x^(2*n-6)*(x^n -y^2)`.

Note that `Delta` is also a cusp form for `n=infinity`.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.graded_ring import␣
↪QuasiMeromorphicModularFormsRing, CuspFormsRing
sage: MR = CuspFormsRing(n=7)
sage: Delta = MR.Delta()
sage: Delta in MR
True
sage: Delta
f_rho^15*d - f_rho^8*f_i^2*d
sage: QuasiMeromorphicModularFormsRing(n=7).Delta() ==␣
↪QuasiMeromorphicModularFormsRing(n=7)(Delta)
True

sage: from sage.modular.modform_hecketriangle.space import CuspForms,␣
↪ModularForms
sage: MF = CuspForms(n=5, k=12)
sage: Delta = MF.Delta()
```

```
sage: Delta in MF
True
sage: CuspFormsRing(n=5, red_hom=True).Delta() == Delta
True
sage: CuspForms(n=5, k=0).Delta() == Delta
True
sage: MF.disp_prec(3)
sage: Delta
q + 47/(200*d)*q^2 + O(q^3)

sage: d = ModularForms(n=5).get_d()
sage: Delta == (d*(ModularForms(n=5).E4()^3-ModularForms(n=5).E6()^2))
True

sage: from sage.modular.modform_hecketriangle.series_constructor import␣
↪MFSeriesConstructor as MFC
sage: MF = CuspForms(n=5, k=12)
sage: d = MF.get_d()
sage: q = MF.get_q()
sage: CuspForms(n=5, k=12).Delta().q_expansion(prec=5) == (d*MFC(group=5,␣
↪prec=7).Delta_ZZ()(q/d)).add_bigoh(5)
True
sage: CuspForms(n=infinity, k=12).Delta().q_expansion(prec=5) ==␣
↪(d*MFC(group=infinity, prec=7).Delta_ZZ()(q/d)).add_bigoh(5)
True
sage: CuspForms(n=5, k=12).Delta().q_expansion(fix_d=1, prec=5) ==␣
↪MFC(group=5, prec=7).Delta_ZZ().add_bigoh(5)
True
sage: CuspForms(n=infinity, k=12).Delta().q_expansion(fix_d=1, prec=5) ==␣
↪MFC(group=infinity, prec=7).Delta_ZZ().add_bigoh(5)
True

sage: CuspForms(n=infinity, k=12).Delta()
q + 24*q^2 + 252*q^3 + 1472*q^4 + O(q^5)

sage: CuspForms(k=12).f_inf() == CuspForms(k=12).Delta()
True
sage: CuspForms(k=12).Delta()
q - 24*q^2 + 252*q^3 - 1472*q^4 + O(q^5)
```

E2 ()

Return the normalized quasi holomorphic Eisenstein series of weight 2.

It lies in a (quasi holomorphic) extension of the graded ring of `self`. In case `has_reduce_hom` is `True` it is given as an element of the corresponding space of homogeneous elements.

It is in particular also a generator of the graded ring of `self` and the polynomial variable `z` exactly corresponds to `E2`.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.graded_ring import␣
↪QuasiMeromorphicModularFormsRing, QuasiModularFormsRing, CuspFormsRing
sage: MR = QuasiModularFormsRing(n=7)
sage: E2 = MR.E2()
sage: E2 in MR
True
sage: CuspFormsRing(n=7).E2() == E2
True
```

```
sage: E2
E2
sage: QuasiMeromorphicModularFormsRing(n=7).E2() ==␣
↪QuasiMeromorphicModularFormsRing(n=7)(E2)
True

sage: from sage.modular.modform_hecketriangle.space import QuasiModularForms,␣
↪CuspForms
sage: MF = QuasiModularForms(n=5, k=2)
sage: E2 = MF.E2()
sage: E2 in MF
True
sage: QuasiModularFormsRing(n=5, red_hom=True).E2() == E2
True
sage: CuspForms(n=5, k=12, ep=1).E2() == E2
True
sage: MF.disp_prec(3)
sage: E2
1 - 9/(200*d)*q - 369/(320000*d^2)*q^2 + O(q^3)

sage: f_inf = MF.f_inf()
sage: E2 == f_inf.derivative() / f_inf
True

sage: from sage.modular.modform_hecketriangle.series_constructor import␣
↪MFSeriesConstructor as MFC
sage: MF = QuasiModularForms(n=5, k=2)
sage: d = MF.get_d()
sage: q = MF.get_q()
sage: QuasiModularForms(n=5, k=2).E2().q_expansion(prec=5) == MFC(group=5,␣
↪prec=7).E2_ZZ()(q/d).add_bigoh(5)
True
sage: QuasiModularForms(n=infinity, k=2).E2().q_expansion(prec=5) ==␣
↪MFC(group=infinity, prec=7).E2_ZZ()(q/d).add_bigoh(5)
True
sage: QuasiModularForms(n=5, k=2).E2().q_expansion(fix_d=1, prec=5) ==␣
↪MFC(group=5, prec=7).E2_ZZ().add_bigoh(5)
True
sage: QuasiModularForms(n=infinity, k=2).E2().q_expansion(fix_d=1, prec=5) ==␣
↪MFC(group=infinity, prec=7).E2_ZZ().add_bigoh(5)
True

sage: QuasiModularForms(n=infinity, k=2).E2()
1 - 8*q - 8*q^2 - 32*q^3 - 40*q^4 + O(q^5)

sage: QuasiModularForms(k=2).E2()
1 - 24*q - 72*q^2 - 96*q^3 - 168*q^4 + O(q^5)
```

E4 ( )

Return the normalized Eisenstein series of weight 4.

It lies in a (holomorphic) extension of the graded ring of `self`. In case `has_reduce_hom` is `True` it is given as an element of the corresponding space of homogeneous elements.

It is equal to `f_rho^(n-2)`.

NOTE:

If `n=infinity` the situation is different, there we have: `f_rho=1` (since that's the limit as `n` goes to infinity) and the polynomial variable `x` refers to `E4` instead of `f_rho`. In that case `E4` has exactly one simple zero at the cusp `-1`. Also note that `E4` is the limit of `f_rho^n`.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.graded_ring import␣
↪QuasiMeromorphicModularFormsRing, ModularFormsRing, CuspFormsRing
sage: MR = ModularFormsRing(n=7)
sage: E4 = MR.E4()
sage: E4 in MR
True
sage: CuspFormsRing(n=7).E4() == E4
True
sage: E4
f_rho^5
sage: QuasiMeromorphicModularFormsRing(n=7).E4() ==␣
↪QuasiMeromorphicModularFormsRing(n=7)(E4)
True

sage: from sage.modular.modform_hecketriangle.space import ModularForms,␣
↪CuspForms
sage: MF = ModularForms(n=5, k=4)
sage: E4 = MF.E4()
sage: E4 in MF
True
sage: ModularFormsRing(n=5, red_hom=True).E4() == E4
True
sage: CuspForms(n=5, k=12).E4() == E4
True
sage: MF.disp_prec(3)
sage: E4
1 + 21/(100*d)*q + 483/(32000*d^2)*q^2 + O(q^3)

sage: from sage.modular.modform_hecketriangle.series_constructor import␣
↪MFSeriesConstructor as MFC
sage: MF = ModularForms(n=5)
sage: d = MF.get_d()
sage: q = MF.get_q()
sage: ModularForms(n=5, k=4).E4().q_expansion(prec=5) == MFC(group=5, prec=7).
↪E4_ZZ()(q/d).add_bigoh(5)
True
sage: ModularForms(n=infinity, k=4).E4().q_expansion(prec=5) ==␣
↪MFC(group=infinity, prec=7).E4_ZZ()(q/d).add_bigoh(5)
True
sage: ModularForms(n=5, k=4).E4().q_expansion(fix_d=1, prec=5) ==␣
↪MFC(group=5, prec=7).E4_ZZ().add_bigoh(5)
True
sage: ModularForms(n=infinity, k=4).E4().q_expansion(fix_d=1, prec=5) ==␣
↪MFC(group=infinity, prec=7).E4_ZZ().add_bigoh(5)
True

sage: ModularForms(n=infinity, k=4).E4()
1 + 16*q + 112*q^2 + 448*q^3 + 1136*q^4 + O(q^5)

sage: ModularForms(k=4).f_rho() == ModularForms(k=4).E4()
True
sage: ModularForms(k=4).E4()
1 + 240*q + 2160*q^2 + 6720*q^3 + 17520*q^4 + O(q^5)
```

E6 ( )

Return the normalized Eisenstein series of weight 6.

It lies in a (holomorphic) extension of the graded ring of `self`. In case `has_reduce_hom` is `True` it is given as an element of the corresponding space of homogeneous elements.

It is equal to `f_rho^(n-3) * f_i`.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.graded_ring import␣
↪QuasiMeromorphicModularFormsRing, ModularFormsRing, CuspFormsRing
sage: MR = ModularFormsRing(n=7)
sage: E6 = MR.E6()
sage: E6 in MR
True
sage: CuspFormsRing(n=7).E6() == E6
True
sage: E6
f_rho^4*f_i
sage: QuasiMeromorphicModularFormsRing(n=7).E6() ==␣
↪QuasiMeromorphicModularFormsRing(n=7)(E6)
True

sage: from sage.modular.modform_hecketriangle.space import ModularForms,␣
↪CuspForms
sage: MF = ModularForms(n=5, k=6)
sage: E6 = MF.E6()
sage: E6 in MF
True
sage: ModularFormsRing(n=5, red_hom=True).E6() == E6
True
sage: CuspForms(n=5, k=12).E6() == E6
True
sage: MF.disp_prec(3)
sage: E6
1 - 37/(200*d)*q - 14663/(320000*d^2)*q^2 + O(q^3)

sage: from sage.modular.modform_hecketriangle.series_constructor import␣
↪MFSeriesConstructor as MFC
sage: MF = ModularForms(n=5, k=6)
sage: d = MF.get_d()
sage: q = MF.get_q()
sage: ModularForms(n=5, k=6).E6().q_expansion(prec=5) == MFC(group=5, prec=7).
↪E6_ZZ()(q/d).add_bigoh(5)
True
sage: ModularForms(n=infinity, k=6).E6().q_expansion(prec=5) ==␣
↪MFC(group=infinity, prec=7).E6_ZZ()(q/d).add_bigoh(5)
True
sage: ModularForms(n=5, k=6).E6().q_expansion(fix_d=1, prec=5) ==␣
↪MFC(group=5, prec=7).E6_ZZ().add_bigoh(5)
True
sage: ModularForms(n=infinity, k=6).E6().q_expansion(fix_d=1, prec=5) ==␣
↪MFC(group=infinity, prec=7).E6_ZZ().add_bigoh(5)
True

sage: ModularForms(n=infinity, k=6).E6()
1 - 8*q - 248*q^2 - 1952*q^3 - 8440*q^4 + O(q^5)

sage: ModularForms(k=6).f_i() == ModularForms(k=6).E6()
```

```
True
sage: ModularForms(k=6).E6()
1 - 504*q - 16632*q^2 - 122976*q^3 - 532728*q^4 + O(q^5)
```

EisensteinSeries (*k=None*)

Return the normalized Eisenstein series of weight `k`.

Only arithmetic groups or trivial weights (with corresponding one dimensional spaces) are supported.

INPUT:

- `k` – A non-negative even integer, namely the weight.

  If `k=None` (default) then the weight of `self` is choosen if `self` is homogeneous and the weight is possible, otherwise `k=0` is set.

OUTPUT:

A modular form element lying in a (holomorphic) extension of the graded ring of `self`. In case `has_reduce_hom` is `True` it is given as an element of the corresponding space of homogeneous elements.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.graded_ring import␣
↪ModularFormsRing, CuspFormsRing
sage: MR = ModularFormsRing()
sage: MR.EisensteinSeries() == MR.one()
True
sage: E8 = MR.EisensteinSeries(k=8)
sage: E8 in MR
True
sage: E8
f_rho^2

sage: from sage.modular.modform_hecketriangle.space import CuspForms,␣
↪ModularForms
sage: MF = ModularForms(n=4, k=12)
sage: E12 = MF.EisensteinSeries()
sage: E12 in MF
True
sage: CuspFormsRing(n=4, red_hom=True).EisensteinSeries(k=12).parent()
ModularForms(n=4, k=12, ep=1) over Integer Ring
sage: MF.disp_prec(4)
sage: E12
1 + 1008/691*q + 2129904/691*q^2 + 178565184/691*q^3 + O(q^4)

sage: from sage.modular.modform_hecketriangle.series_constructor import␣
↪MFSeriesConstructor as MFC
sage: d = MF.get_d()
sage: q = MF.get_q()
sage: ModularForms(n=3, k=2).EisensteinSeries().q_expansion(prec=5) ==␣
↪MFC(group=3, prec=7).EisensteinSeries_ZZ(k=2)(q/d).add_bigoh(5)
True
sage: ModularForms(n=3, k=4).EisensteinSeries().q_expansion(prec=5) ==␣
↪MFC(group=3, prec=7).EisensteinSeries_ZZ(k=4)(q/d).add_bigoh(5)
True
sage: ModularForms(n=3, k=6).EisensteinSeries().q_expansion(prec=5) ==␣
↪MFC(group=3, prec=7).EisensteinSeries_ZZ(k=6)(q/d).add_bigoh(5)
True
```

```
sage: ModularForms(n=3, k=8).EisensteinSeries().q_expansion(prec=5) ==␣
↪MFC(group=3, prec=7).EisensteinSeries_ZZ(k=8)(q/d).add_bigoh(5)
True
sage: ModularForms(n=4, k=2).EisensteinSeries().q_expansion(prec=5) ==␣
↪MFC(group=4, prec=7).EisensteinSeries_ZZ(k=2)(q/d).add_bigoh(5)
True
sage: ModularForms(n=4, k=4).EisensteinSeries().q_expansion(prec=5) ==␣
↪MFC(group=4, prec=7).EisensteinSeries_ZZ(k=4)(q/d).add_bigoh(5)
True
sage: ModularForms(n=4, k=6).EisensteinSeries().q_expansion(prec=5) ==␣
↪MFC(group=4, prec=7).EisensteinSeries_ZZ(k=6)(q/d).add_bigoh(5)
True
sage: ModularForms(n=4, k=8).EisensteinSeries().q_expansion(prec=5) ==␣
↪MFC(group=4, prec=7).EisensteinSeries_ZZ(k=8)(q/d).add_bigoh(5)
True
sage: ModularForms(n=6, k=2, ep=-1).EisensteinSeries().q_expansion(prec=5) ==␣
↪MFC(group=6, prec=7).EisensteinSeries_ZZ(k=2)(q/d).add_bigoh(5)
True
sage: ModularForms(n=6, k=4).EisensteinSeries().q_expansion(prec=5) ==␣
↪MFC(group=6, prec=7).EisensteinSeries_ZZ(k=4)(q/d).add_bigoh(5)
True
sage: ModularForms(n=6, k=6, ep=-1).EisensteinSeries().q_expansion(prec=5) ==␣
↪MFC(group=6, prec=7).EisensteinSeries_ZZ(k=6)(q/d).add_bigoh(5)
True
sage: ModularForms(n=6, k=8).EisensteinSeries().q_expansion(prec=5) ==␣
↪MFC(group=6, prec=7).EisensteinSeries_ZZ(k=8)(q/d).add_bigoh(5)
True

sage: ModularForms(n=3, k=12).EisensteinSeries()
1 + 65520/691*q + 134250480/691*q^2 + 11606736960/691*q^3 + 274945048560/
↪691*q^4 + O(q^5)
sage: ModularForms(n=4, k=12).EisensteinSeries()
1 + 1008/691*q + 2129904/691*q^2 + 178565184/691*q^3 + O(q^4)
sage: ModularForms(n=6, k=12).EisensteinSeries()
1 + 6552/50443*q + 13425048/50443*q^2 + 1165450104/50443*q^3 + 27494504856/
↪50443*q^4 + O(q^5)
sage: ModularForms(n=3, k=20).EisensteinSeries()
1 + 13200/174611*q + 6920614800/174611*q^2 + 15341851377600/174611*q^3 +␣
↪3628395292275600/174611*q^4 + O(q^5)
sage: ModularForms(n=4).EisensteinSeries(k=8)
1 + 480/17*q + 69600/17*q^2 + 1050240/17*q^3 + 8916960/17*q^4 + O(q^5)
sage: ModularForms(n=6).EisensteinSeries(k=20)
1 + 264/206215591*q + 138412296/206215591*q^2 + 306852616488/206215591*q^3 +␣
↪72567905845512/206215591*q^4 + O(q^5)
```

Element

alias of *FormsRingElement*

FormsRingElement

alias of *FormsRingElement*

G_inv ()

If 2 divides n: Return the G-invariant of the group of `self`.

The G-invariant is analogous to the J-invariant but has multiplier -1. I.e. `G_inv(-1/t) = -G_inv(t)`. It is a holomorphic square root of `J_inv*(J_inv-1)` with real Fourier coefficients.

If 2 does not divide n the function does not exist and an exception is raised.

The G-invariant lies in a (weak) extension of the graded ring of `self`. In case `has_reduce_hom` is `True` it is given as an element of the corresponding space of homogeneous elements.

NOTE:

If `n=infinity` then `G_inv` is holomorphic everywhere except at the cusp `-1` where it isn't even meromorphic. Consequently this function raises an exception for `n=infinity`.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.graded_ring import␣
↪QuasiMeromorphicModularFormsRing, WeakModularFormsRing, CuspFormsRing
sage: MR = WeakModularFormsRing(n=8)
sage: G_inv = MR.G_inv()
sage: G_inv in MR
True
sage: CuspFormsRing(n=8).G_inv() == G_inv
True
sage: G_inv
f_rho^4*f_i*d/(f_rho^8 - f_i^2)
sage: QuasiMeromorphicModularFormsRing(n=8).G_inv() ==␣
↪QuasiMeromorphicModularFormsRing(n=8)(G_inv)
True

sage: from sage.modular.modform_hecketriangle.space import WeakModularForms,␣
↪CuspForms
sage: MF = WeakModularForms(n=8, k=0, ep=-1)
sage: G_inv = MF.G_inv()
sage: G_inv in MF
True
sage: WeakModularFormsRing(n=8, red_hom=True).G_inv() == G_inv
True
sage: CuspForms(n=8, k=12, ep=1).G_inv() == G_inv
True
sage: MF.disp_prec(3)
sage: G_inv
d^2*q^-1 - 15*d/128 - 15139/262144*q - 11575/(1572864*d)*q^2 + O(q^3)

sage: from sage.modular.modform_hecketriangle.series_constructor import␣
↪MFSeriesConstructor as MFC
sage: MF = WeakModularForms(n=8)
sage: d = MF.get_d()
sage: q = MF.get_q()
sage: WeakModularForms(n=8).G_inv().q_expansion(prec=5) == (d*MFC(group=8,␣
↪prec=7).G_inv_ZZ()(q/d)).add_bigoh(5)
True
sage: WeakModularForms(n=8).G_inv().q_expansion(fix_d=1, prec=5) ==␣
↪MFC(group=8, prec=7).G_inv_ZZ().add_bigoh(5)
True

sage: WeakModularForms(n=4, k=0, ep=-1).G_inv()
1/65536*q^-1 - 3/8192 - 955/16384*q - 49/32*q^2 - 608799/32768*q^3 - 659/4*q^
↪4 + O(q^5)

As explained above, the G-invariant exists only for even `n`::

sage: from sage.modular.modform_hecketriangle.space import WeakModularForms
sage: MF = WeakModularForms(n=9)
sage: MF.G_inv()
Traceback (most recent call last):
```

```
...
ArithmeticError: G_inv doesn't exist for odd n(=9).
```

J_inv ()

Return the J-invariant (Hauptmodul) of the group of `self`. It is normalized such that `J_inv(infinity) = infinity`, it has real Fourier coefficients starting with `d > 0` and `J_inv(i) = 1`

It lies in a (weak) extension of the graded ring of `self`. In case `has_reduce_hom` is `True` it is given as an element of the corresponding space of homogeneous elements.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.graded_ring import␣
↪QuasiMeromorphicModularFormsRing, WeakModularFormsRing, CuspFormsRing
sage: MR = WeakModularFormsRing(n=7)
sage: J_inv = MR.J_inv()
sage: J_inv in MR
True
sage: CuspFormsRing(n=7).J_inv() == J_inv
True
sage: J_inv
f_rho^7/(f_rho^7 - f_i^2)
sage: QuasiMeromorphicModularFormsRing(n=7).J_inv() ==␣
↪QuasiMeromorphicModularFormsRing(n=7)(J_inv)
True

sage: from sage.modular.modform_hecketriangle.space import WeakModularForms,␣
↪CuspForms
sage: MF = WeakModularForms(n=5, k=0)
sage: J_inv = MF.J_inv()
sage: J_inv in MF
True
sage: WeakModularFormsRing(n=5, red_hom=True).J_inv() == J_inv
True
sage: CuspForms(n=5, k=12).J_inv() == J_inv
True
sage: MF.disp_prec(3)
sage: J_inv
d*q^-1 + 79/200 + 42877/(640000*d)*q + 12957/(2000000*d^2)*q^2 + O(q^3)

sage: from sage.modular.modform_hecketriangle.series_constructor import␣
↪MFSeriesConstructor as MFC
sage: MF = WeakModularForms(n=5)
sage: d = MF.get_d()
sage: q = MF.get_q()
sage: WeakModularForms(n=5).J_inv().q_expansion(prec=5) == MFC(group=5,␣
↪prec=7).J_inv_ZZ()(q/d).add_bigoh(5)
True
sage: WeakModularForms(n=infinity).J_inv().q_expansion(prec=5) ==␣
↪MFC(group=infinity, prec=7).J_inv_ZZ()(q/d).add_bigoh(5)
True
sage: WeakModularForms(n=5).J_inv().q_expansion(fix_d=1, prec=5) ==␣
↪MFC(group=5, prec=7).J_inv_ZZ().add_bigoh(5)
True
sage: WeakModularForms(n=infinity).J_inv().q_expansion(fix_d=1, prec=5) ==␣
↪MFC(group=infinity, prec=7).J_inv_ZZ().add_bigoh(5)
True
```

```
sage: WeakModularForms(n=infinity).J_inv()
1/64*q^-1 + 3/8 + 69/16*q + 32*q^2 + 5601/32*q^3 + 768*q^4 + O(q^5)

sage: WeakModularForms().J_inv()
1/1728*q^-1 + 31/72 + 1823/16*q + 335840/27*q^2 + 16005555/32*q^3 +␣
↪11716352*q^4 + O(q^5)
```

analytic_type ( )

Return the analytic type of `self`.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.graded_ring import␣
↪QuasiMeromorphicModularFormsRing, QuasiWeakModularFormsRing
sage: QuasiMeromorphicModularFormsRing().analytic_type()
quasi meromorphic modular
sage: QuasiWeakModularFormsRing().analytic_type()
quasi weakly holomorphic modular

sage: from sage.modular.modform_hecketriangle.space import␣
↪MeromorphicModularForms, CuspForms
sage: MeromorphicModularForms(k=10).analytic_type()
meromorphic modular
sage: CuspForms(n=7, k=12, base_ring=AA).analytic_type()
cuspidal
```

base_ring ( )

Return base ring of `self`.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.graded_ring import␣
↪ModularFormsRing
sage: ModularFormsRing().base_ring()
Integer Ring

sage: from sage.modular.modform_hecketriangle.space import CuspForms
sage: CuspForms(k=12, base_ring=AA).base_ring()
Algebraic Real Field
```

change_ring (*new_base_ring*)

Return the same space as `self` but over a new base ring `new_base_ring`.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.graded_ring import␣
↪ModularFormsRing
sage: ModularFormsRing().change_ring(CC)
ModularFormsRing(n=3) over Complex Field with 53 bits of precision
```

coeff_ring ( )

Return coefficient ring of `self`.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.graded_ring import␣
↪ModularFormsRing
sage: ModularFormsRing().coeff_ring()
```

```
Fraction Field of Univariate Polynomial Ring in d over Integer Ring

sage: from sage.modular.modform_hecketriangle.space import CuspForms
sage: CuspForms(k=12, base_ring=AA).coeff_ring()
Fraction Field of Univariate Polynomial Ring in d over Algebraic Real Field
```

construction ( )

Return a functor that constructs `self` (used by the coercion machinery).

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.graded_ring import␣
↪ModularFormsRing
sage: ModularFormsRing().construction()
(ModularFormsRingFunctor(n=3), BaseFacade(Integer Ring))
```

contains_coeff_ring ( )

Return whether `self` contains its coefficient ring.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.graded_ring import␣
↪CuspFormsRing, ModularFormsRing
sage: CuspFormsRing(n=4).contains_coeff_ring()
False
sage: ModularFormsRing(n=5).contains_coeff_ring()
True
```

default_num_prec (*prec=None*)

Set the default numerical precision to `prec` (default: `53`). If `prec=None` (default) the current default numerical precision is returned instead.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.space import ModularForms
sage: MF = ModularForms(k=6)
sage: MF.default_prec(20)
sage: MF.default_num_prec(10)
sage: MF.default_num_prec()
10
sage: E6 = MF.E6()
sage: E6(i + 1e-1000)
0.002... - 6.7...e-1000*I
sage: MF.default_num_prec(100)
sage: E6(i + 1e-1000)
3.9946838...e-1999 - 6.6578064...e-1000*I

sage: MF = ModularForms(n=5, k=4/3)
sage: f_rho = MF.f_rho()
sage: f_rho.q_expansion(prec=2)[1]
7/(100*d)
sage: MF.default_num_prec(15)
sage: f_rho.q_expansion_fixed_d(prec=2)[1]
9.9...
sage: MF.default_num_prec(100)
sage: f_rho.q_expansion_fixed_d(prec=2)[1]
9.92593243510795915276017782...
```

default_prec (*prec=None*)

Set the default precision `prec` for the Fourier expansion. If `prec=None` (default) then the current default precision is returned instead.

INPUT:

- `prec` – An integer.

NOTE:

This is also used as the default precision for the Fourier expansion when evaluating forms.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.graded_ring import␣
↪ModularFormsRing
sage: from sage.modular.modform_hecketriangle.space import ModularForms
sage: MR = ModularFormsRing()
sage: MR.default_prec(3)
sage: MR.default_prec()
3
sage: MR.Delta().q_expansion_fixed_d()
q - 24*q^2 + O(q^3)
sage: MF = ModularForms(k=4)
sage: MF.default_prec(2)
sage: MF.E4()
1 + 240*q + O(q^2)
sage: MF.default_prec()
2
```

diff_alg ( )

Return the algebra of differential operators (over QQ) which is used on rational functions representing elements of `self`.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.graded_ring import␣
↪ModularFormsRing
sage: ModularFormsRing().diff_alg()
Noncommutative Multivariate Polynomial Ring in X, Y, Z, dX, dY, dZ over␣
↪Rational Field, nc-relations: {dZ*Z: Z*dZ + 1, dY*Y: Y*dY + 1, dX*X: X*dX +␣
↪1}

sage: from sage.modular.modform_hecketriangle.space import CuspForms
sage: CuspForms(k=12, base_ring=AA).diff_alg()
Noncommutative Multivariate Polynomial Ring in X, Y, Z, dX, dY, dZ over␣
↪Rational Field, nc-relations: {dZ*Z: Z*dZ + 1, dY*Y: Y*dY + 1, dX*X: X*dX +␣
↪1}
```

disp_prec (*prec=None*)

Set the maximal display precision to `prec`. If `prec="max"` the precision is set to the default precision. If `prec=None` (default) then the current display precision is returned instead.

NOTE:

This is used for displaying/representing (elements of) `self` as Fourier expansions.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.space import ModularForms
sage: MF = ModularForms(k=4)
```

```
sage: MF.default_prec(5)
sage: MF.disp_prec(3)
sage: MF.disp_prec()
3
sage: MF.E4()
1 + 240*q + 2160*q^2 + O(q^3)
sage: MF.disp_prec("max")
sage: MF.E4()
1 + 240*q + 2160*q^2 + 6720*q^3 + 17520*q^4 + O(q^5)
```

extend_type (*analytic_type=None*, *ring=False*)

Return a new space which contains (elements of) `self` with the analytic type of `self` extended by `analytic_type`, possibly extended to a graded ring in case `ring` is `True`.

INPUT:

- **`analytic_type` – An `AnalyticType` or something which** coerces into it (default: `None`).
- `ring` – Whether to extend to a graded ring (default: `False`).

OUTPUT:

The new extended space.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.graded_ring import␣
↪ModularFormsRing
sage: from sage.modular.modform_hecketriangle.space import CuspForms

sage: MR = ModularFormsRing(n=5)
sage: MR.extend_type(["quasi", "weak"])
QuasiWeakModularFormsRing(n=5) over Integer Ring

sage: CF=CuspForms(k=12)
sage: CF.extend_type("holo")
ModularForms(n=3, k=12, ep=1) over Integer Ring
sage: CF.extend_type("quasi", ring=True)
QuasiCuspFormsRing(n=3) over Integer Ring

sage: CF.subspace([CF.Delta()]).extend_type()
CuspForms(n=3, k=12, ep=1) over Integer Ring
```

f_i ( )

Return a normalized modular form `f_i` with exactly one simple zero at `i` (up to the group action).

It lies in a (holomorphic) extension of the graded ring of `self`. In case `has_reduce_hom` is `True` it is given as an element of the corresponding space of homogeneous elements.

The polynomial variable `y` exactly corresponds to `f_i`.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.graded_ring import␣
↪QuasiMeromorphicModularFormsRing, ModularFormsRing, CuspFormsRing
sage: MR = ModularFormsRing(n=7)
sage: f_i = MR.f_i()
sage: f_i in MR
True
sage: CuspFormsRing(n=7).f_i() == f_i
True
```

```
sage: f_i
f_i
sage: QuasiMeromorphicModularFormsRing(n=7).f_i() ==␣
↪QuasiMeromorphicModularFormsRing(n=7)(f_i)
True

sage: from sage.modular.modform_hecketriangle.space import ModularForms,␣
↪CuspForms
sage: MF = ModularForms(n=5, k=10/3)
sage: f_i = MF.f_i()
sage: f_i in MF
True
sage: ModularFormsRing(n=5, red_hom=True).f_i() == f_i
True
sage: CuspForms(n=5, k=12).f_i() == f_i
True
sage: MF.disp_prec(3)
sage: f_i
1 - 13/(40*d)*q - 351/(64000*d^2)*q^2 + O(q^3)

sage: from sage.modular.modform_hecketriangle.series_constructor import␣
↪MFSeriesConstructor as MFC
sage: MF = ModularForms(n=5)
sage: d = MF.get_d()
sage: q = MF.get_q()
sage: ModularForms(n=5).f_i().q_expansion(prec=5) == MFC(group=5, prec=7).f_i_
↪ZZ()(q/d).add_bigoh(5)
True
sage: ModularForms(n=infinity).f_i().q_expansion(prec=5) ==␣
↪MFC(group=infinity, prec=7).f_i_ZZ()(q/d).add_bigoh(5)
True
sage: ModularForms(n=5).f_i().q_expansion(fix_d=1, prec=5) == MFC(group=5,␣
↪prec=7).f_i_ZZ().add_bigoh(5)
True
sage: ModularForms(n=infinity).f_i().q_expansion(fix_d=1, prec=5) ==␣
↪MFC(group=infinity, prec=7).f_i_ZZ().add_bigoh(5)
True

sage: ModularForms(n=infinity, k=2).f_i()
1 - 24*q + 24*q^2 - 96*q^3 + 24*q^4 + O(q^5)

sage: ModularForms(k=6).f_i() == ModularForms(k=4).E6()
True
sage: ModularForms(k=6).f_i()
1 - 504*q - 16632*q^2 - 122976*q^3 - 532728*q^4 + O(q^5)
```

f_inf ( )

Return a normalized (according to its first nontrivial Fourier coefficient) cusp form `f_inf` with exactly one simple zero at `infinity` (up to the group action).

It lies in a (cuspidal) extension of the graded ring of `self`. In case `has_reduce_hom` is `True` it is given as an element of the corresponding space of homogeneous elements.

NOTE:

If `n=infinity` then `f_inf` is no longer a cusp form since it doesn't vanish at the cusp `-1`. The first non-trivial cusp form is given by `E4*f_inf`.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.graded_ring import␣
↪QuasiMeromorphicModularFormsRing, CuspFormsRing
sage: MR = CuspFormsRing(n=7)
sage: f_inf = MR.f_inf()
sage: f_inf in MR
True
sage: f_inf
f_rho^7*d - f_i^2*d
sage: QuasiMeromorphicModularFormsRing(n=7).f_inf() ==␣
↪QuasiMeromorphicModularFormsRing(n=7)(f_inf)
True

sage: from sage.modular.modform_hecketriangle.space import CuspForms,␣
↪ModularForms
sage: MF = CuspForms(n=5, k=20/3)
sage: f_inf = MF.f_inf()
sage: f_inf in MF
True
sage: CuspFormsRing(n=5, red_hom=True).f_inf() == f_inf
True
sage: CuspForms(n=5, k=0).f_inf() == f_inf
True
sage: MF.disp_prec(3)
sage: f_inf
q - 9/(200*d)*q^2 + O(q^3)

sage: from sage.modular.modform_hecketriangle.series_constructor import␣
↪MFSeriesConstructor as MFC
sage: MF = ModularForms(n=5)
sage: d = MF.get_d()
sage: q = MF.get_q()
sage: ModularForms(n=5).f_inf().q_expansion(prec=5) == (d*MFC(group=5,␣
↪prec=7).f_inf_ZZ()(q/d)).add_bigoh(5)
True
sage: ModularForms(n=infinity).f_inf().q_expansion(prec=5) ==␣
↪(d*MFC(group=infinity, prec=7).f_inf_ZZ()(q/d)).add_bigoh(5)
True
sage: ModularForms(n=5).f_inf().q_expansion(fix_d=1, prec=5) == MFC(group=5,␣
↪prec=7).f_inf_ZZ().add_bigoh(5)
True
sage: ModularForms(n=infinity).f_inf().q_expansion(fix_d=1, prec=5) ==␣
↪MFC(group=infinity, prec=7).f_inf_ZZ().add_bigoh(5)
True

sage: ModularForms(n=infinity, k=4).f_inf().reduced_parent()
ModularForms(n=+Infinity, k=4, ep=1) over Integer Ring
sage: ModularForms(n=infinity, k=4).f_inf()
q - 8*q^2 + 28*q^3 - 64*q^4 + O(q^5)

sage: CuspForms(k=12).f_inf() == CuspForms(k=12).Delta()
True
sage: CuspForms(k=12).f_inf()
q - 24*q^2 + 252*q^3 - 1472*q^4 + O(q^5)
```

f_rho ( )

Return a normalized modular form `f_rho` with exactly one simple zero at `rho` (up to the group action).

It lies in a (holomorphic) extension of the graded ring of `self`. In case `has_reduce_hom` is `True` it is given as an element of the corresponding space of homogeneous elements.

The polynomial variable `x` exactly corresponds to `f_rho`.

NOTE:

If `n=infinity` the situation is different, there we have: `f_rho=1` (since that's the limit as `n` goes to infinity) and the polynomial variable `x` no longer refers to `f_rho`. Instead it refers to `E4` which has exactly one simple zero at the cusp `-1`. Also note that `E4` is the limit of `f_rho^(n-2)`.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.graded_ring import␣
↪QuasiMeromorphicModularFormsRing, ModularFormsRing, CuspFormsRing
sage: MR = ModularFormsRing(n=7)
sage: f_rho = MR.f_rho()
sage: f_rho in MR
True
sage: CuspFormsRing(n=7).f_rho() == f_rho
True
sage: f_rho
f_rho
sage: QuasiMeromorphicModularFormsRing(n=7).f_rho() ==␣
↪QuasiMeromorphicModularFormsRing(n=7)(f_rho)
True

sage: from sage.modular.modform_hecketriangle.space import ModularForms,␣
↪CuspForms
sage: MF = ModularForms(n=5, k=4/3)
sage: f_rho = MF.f_rho()
sage: f_rho in MF
True
sage: ModularFormsRing(n=5, red_hom=True).f_rho() == f_rho
True
sage: CuspForms(n=5, k=12).f_rho() == f_rho
True
sage: MF.disp_prec(3)
sage: f_rho
1 + 7/(100*d)*q + 21/(160000*d^2)*q^2 + O(q^3)

sage: from sage.modular.modform_hecketriangle.series_constructor import␣
↪MFSeriesConstructor as MFC
sage: MF = ModularForms(n=5)
sage: d = MF.get_d()
sage: q = MF.get_q()
sage: ModularForms(n=5).f_rho().q_expansion(prec=5) == MFC(group=5, prec=7).f_
↪rho_ZZ()(q/d).add_bigoh(5)
True
sage: ModularForms(n=infinity).f_rho().q_expansion(prec=5) ==␣
↪MFC(group=infinity, prec=7).f_rho_ZZ()(q/d).add_bigoh(5)
True
sage: ModularForms(n=5).f_rho().q_expansion(fix_d=1, prec=5) == MFC(group=5,␣
↪prec=7).f_rho_ZZ().add_bigoh(5)
True
sage: ModularForms(n=infinity).f_rho().q_expansion(fix_d=1, prec=5) ==␣
↪MFC(group=infinity, prec=7).f_rho_ZZ().add_bigoh(5)
True

sage: ModularForms(n=infinity, k=0).f_rho() == ModularForms(n=infinity,␣
↪k=0)(1)
```

```
True

sage: ModularForms(k=4).f_rho() == ModularForms(k=4).E4()
True
sage: ModularForms(k=4).f_rho()
1 + 240*q + 2160*q^2 + 6720*q^3 + 17520*q^4 + O(q^5)
```

g_inv ( )

If 2 divides n: Return the g-invariant of the group of `self`.

The g-invariant is analogous to the j-invariant but has multiplier `-1`. I.e. `g_inv(-1/t) = -g_inv(t)`. It is a (normalized) holomorphic square root of `J_inv*(J_inv-1)`, normalized such that its first nontrivial Fourier coefficient is `1`.

If 2 does not divide `n` the function does not exist and an exception is raised.

The g-invariant lies in a (weak) extension of the graded ring of `self`. In case `has_reduce_hom` is `True` it is given as an element of the corresponding space of homogeneous elements.

NOTE:

If `n=infinity` then `g_inv` is holomorphic everywhere except at the cusp `-1` where it isn't even meromorphic. Consequently this function raises an exception for `n=infinity`.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.graded_ring import␣
↪QuasiMeromorphicModularFormsRing, WeakModularFormsRing, CuspFormsRing
sage: MR = WeakModularFormsRing(n=8)
sage: g_inv = MR.g_inv()
sage: g_inv in MR
True
sage: CuspFormsRing(n=8).g_inv() == g_inv
True
sage: g_inv
f_rho^4*f_i/(f_rho^8*d - f_i^2*d)
sage: QuasiMeromorphicModularFormsRing(n=8).g_inv() ==␣
↪QuasiMeromorphicModularFormsRing(n=8)(g_inv)
True

sage: from sage.modular.modform_hecketriangle.space import WeakModularForms,␣
↪CuspForms
sage: MF = WeakModularForms(n=8, k=0, ep=-1)
sage: g_inv = MF.g_inv()
sage: g_inv in MF
True
sage: WeakModularFormsRing(n=8, red_hom=True).g_inv() == g_inv
True
sage: CuspForms(n=8, k=12, ep=1).g_inv() == g_inv
True
sage: MF.disp_prec(3)
sage: g_inv
q^-1 - 15/(128*d) - 15139/(262144*d^2)*q - 11575/(1572864*d^3)*q^2 + O(q^3)

sage: WeakModularForms(n=4, k=0, ep=-1).g_inv()
q^-1 - 24 - 3820*q - 100352*q^2 - 1217598*q^3 - 10797056*q^4 + O(q^5)

As explained above, the g-invariant exists only for even `n`::

sage: from sage.modular.modform_hecketriangle.space import WeakModularForms
```

```
sage: MF = WeakModularForms(n=9)
sage: MF.g_inv()
Traceback (most recent call last):
...
ArithmeticError: g_inv doesn't exist for odd n(=9).
```

get_d (*fix_d=False*, *d_num_prec=None*)

Return the parameter `d` of self either as a formal parameter or as a numerical approximation with the specified precision (resp. an exact value in the arithmetic cases).

For an (exact) symbolic expression also see `HeckeTriangleGroup().dvalue()`.

INPUT:

- **`fix_d` – If `False` (default) a formal parameter is** used for `d`.

  If `True` then the numerical value of `d` is used (or an exact value if the group is arithmetic). Otherwise, the given value is used for `d`.

- **`d_num_prec` – An integer. The numerical precision of** `d`. Default: `None`, in which case the default numerical precision of `self.parent()` is used.

OUTPUT:

The corresponding formal, numerical or exact parameter `d` of `self`, depending on the arguments and whether `self.group()` is arithmetic.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.graded_ring import␣
↪ModularFormsRing
sage: ModularFormsRing(n=8).get_d()
d
sage: ModularFormsRing(n=8).get_d().parent()
Fraction Field of Univariate Polynomial Ring in d over Integer Ring
sage: ModularFormsRing(n=infinity).get_d(fix_d = True)
1/64
sage: ModularFormsRing(n=infinity).get_d(fix_d = True).parent()
Rational Field
sage: ModularFormsRing(n=5).default_num_prec(40)
sage: ModularFormsRing(n=5).get_d(fix_d = True)
0.0070522341...
sage: ModularFormsRing(n=5).get_d(fix_d = True).parent()
Real Field with 40 bits of precision
sage: ModularFormsRing(n=5).get_d(fix_d = True, d_num_prec=100).parent()
Real Field with 100 bits of precision
sage: ModularFormsRing(n=5).get_d(fix_d=1).parent()
Integer Ring
```

get_q (*prec=None*, *fix_d=False*, *d_num_prec=None*)

Return the generator of the power series of the Fourier expansion of `self`.

INPUT:

- **`prec` – An integer or `None` (default), namely the desired default** precision of the space of power series. If nothing is specified the default precision of `self` is used.

- **`fix_d` – If `False` (default) a formal parameter is used for `d`.** If `True` then the numerical value of `d` is used (resp. an exact value if the group is arithmetic). Otherwise the given value is used for `d`.

- **d_num_prec – The precision to be used if a numerical value for d is substituted.** Default: `None` in which case the default numerical precision of `self.parent()` is used.

OUTPUT:

The generator of the `PowerSeriesRing` of corresponding to the given parameters. The base ring of the power series ring is given by the corresponding parent of `self.get_d()` with the same arguments.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.graded_ring import␣
↪ModularFormsRing
sage: ModularFormsRing(n=8).default_prec(5)
sage: ModularFormsRing(n=8).get_q().parent()
Power Series Ring in q over Fraction Field of Univariate Polynomial Ring in d␣
↪over Integer Ring
sage: ModularFormsRing(n=8).get_q().parent().default_prec()
5
sage: ModularFormsRing(n=infinity).get_q(prec=12, fix_d = True).parent()
Power Series Ring in q over Rational Field
sage: ModularFormsRing(n=infinity).get_q(prec=12, fix_d = True).parent().
↪default_prec()
12
sage: ModularFormsRing(n=5).default_num_prec(40)
sage: ModularFormsRing(n=5).get_q(fix_d = True).parent()
Power Series Ring in q over Real Field with 40 bits of precision
sage: ModularFormsRing(n=5).get_q(fix_d = True, d_num_prec=100).parent()
Power Series Ring in q over Real Field with 100 bits of precision
sage: ModularFormsRing(n=5).get_q(fix_d=1).parent()
Power Series Ring in q over Rational Field
```

graded_ring ( )

Return the graded ring containing `self`.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.graded_ring import␣
↪ModularFormsRing, CuspFormsRing
sage: from sage.modular.modform_hecketriangle.space import CuspForms

sage: MR = ModularFormsRing(n=5)
sage: MR.graded_ring() == MR
True

sage: CF=CuspForms(k=12)
sage: CF.graded_ring() == CuspFormsRing()
False
sage: CF.graded_ring() == CuspFormsRing(red_hom=True)
True

sage: CF.subspace([CF.Delta()]).graded_ring() == CuspFormsRing(red_hom=True)
True
```

group ( )

Return the (Hecke triangle) group of `self`.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.graded_ring import␣
↪ModularFormsRing
sage: MR = ModularFormsRing(n=7)
sage: MR.group()
Hecke triangle group for n = 7

sage: from sage.modular.modform_hecketriangle.space import CuspForms
sage: CF = CuspForms(n=7, k=4/5)
sage: CF.group()
Hecke triangle group for n = 7
```

has_reduce_hom ( )

Return whether the method `reduce` should reduce homogeneous elements to the corresponding space of homogeneous elements.

This is mainly used by binary operations on homogeneous spaces which temporarily produce an element of `self` but want to consider it as a homogeneous element (also see `reduce`).

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.graded_ring import␣
↪ModularFormsRing
sage: ModularFormsRing().has_reduce_hom()
False
sage: ModularFormsRing(red_hom=True).has_reduce_hom()
True

sage: from sage.modular.modform_hecketriangle.space import ModularForms
sage: ModularForms(k=6).has_reduce_hom()
True
sage: ModularForms(k=6).graded_ring().has_reduce_hom()
True
```

hecke_n ( )

Return the parameter `n` of the (Hecke triangle) group of `self`.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.graded_ring import␣
↪ModularFormsRing
sage: MR = ModularFormsRing(n=7)
sage: MR.hecke_n()
7

sage: from sage.modular.modform_hecketriangle.space import CuspForms
sage: CF = CuspForms(n=7, k=4/5)
sage: CF.hecke_n()
7
```

homogeneous_part (*k*, *ep*)

Return the homogeneous component of degree (`k`, `e`) of `self`.

INPUT:

- `k` – An integer.
- `ep` – $+1$ or -1.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.graded_ring import␣
↪QuasiMeromorphicModularFormsRing, QuasiWeakModularFormsRing
sage: QuasiMeromorphicModularFormsRing(n=7).homogeneous_part(k=2, ep=-1)
QuasiMeromorphicModularForms(n=7, k=2, ep=-1) over Integer Ring
```

is_cuspidal ( )

Return whether `self` only contains cuspidal elements.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.graded_ring import␣
↪QuasiModularFormsRing, QuasiCuspFormsRing
sage: QuasiModularFormsRing().is_cuspidal()
False
sage: QuasiCuspFormsRing().is_cuspidal()
True

sage: from sage.modular.modform_hecketriangle.space import ModularForms,␣
↪QuasiCuspForms
sage: ModularForms(k=12).is_cuspidal()
False
sage: QuasiCuspForms(k=12).is_cuspidal()
True
```

is_holomorphic ( )

Return whether `self` only contains holomorphic modular elements.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.graded_ring import␣
↪QuasiWeakModularFormsRing, QuasiModularFormsRing
sage: QuasiWeakModularFormsRing().is_holomorphic()
False
sage: QuasiModularFormsRing().is_holomorphic()
True

sage: from sage.modular.modform_hecketriangle.space import WeakModularForms,␣
↪CuspForms
sage: WeakModularForms(k=10).is_holomorphic()
False
sage: CuspForms(n=7, k=12, base_ring=AA).is_holomorphic()
True
```

is_homogeneous ( )

Return whether `self` is homogeneous component.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.graded_ring import␣
↪ModularFormsRing
sage: ModularFormsRing().is_homogeneous()
False

sage: from sage.modular.modform_hecketriangle.space import ModularForms
sage: ModularForms(k=6).is_homogeneous()
True
```

is_modular ( )

Return whether `self` only contains modular elements.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.graded_ring import␣
↪QuasiWeakModularFormsRing, CuspFormsRing
sage: QuasiWeakModularFormsRing().is_modular()
False
sage: CuspFormsRing(n=7).is_modular()
True

sage: from sage.modular.modform_hecketriangle.space import␣
↪QuasiWeakModularForms, CuspForms
sage: QuasiWeakModularForms(k=10).is_modular()
False
sage: CuspForms(n=7, k=12, base_ring=AA).is_modular()
True
```

is_weakly_holomorphic ( )

Return whether `self` only contains weakly holomorphic modular elements.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.graded_ring import␣
↪QuasiMeromorphicModularFormsRing, QuasiWeakModularFormsRing, CuspFormsRing
sage: QuasiMeromorphicModularFormsRing().is_weakly_holomorphic()
False
sage: QuasiWeakModularFormsRing().is_weakly_holomorphic()
True

sage: from sage.modular.modform_hecketriangle.space import␣
↪MeromorphicModularForms, CuspForms
sage: MeromorphicModularForms(k=10).is_weakly_holomorphic()
False
sage: CuspForms(n=7, k=12, base_ring=AA).is_weakly_holomorphic()
True
```

is_zerospace ( )

Return whether `self` is the (0-dimensional) zero space.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.graded_ring import␣
↪ModularFormsRing
sage: ModularFormsRing().is_zerospace()
False

sage: from sage.modular.modform_hecketriangle.space import ModularForms,␣
↪CuspForms
sage: ModularForms(k=12).is_zerospace()
False
sage: CuspForms(k=12).reduce_type([]).is_zerospace()
True
```

j_inv ( )

Return the j-invariant (Hauptmodul) of the group of `self` . It is normalized such that `j_inv(infinity) = infinity` , and such that it has real Fourier coefficients starting with `1` .

It lies in a (weak) extension of the graded ring of `self` . In case `has_reduce_hom` is `True` it is given as an element of the corresponding space of homogeneous elements.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.graded_ring import␣
↪QuasiMeromorphicModularFormsRing, WeakModularFormsRing, CuspFormsRing
sage: MR = WeakModularFormsRing(n=7)
sage: j_inv = MR.j_inv()
sage: j_inv in MR
True
sage: CuspFormsRing(n=7).j_inv() == j_inv
True
sage: j_inv
f_rho^7/(f_rho^7*d - f_i^2*d)
sage: QuasiMeromorphicModularFormsRing(n=7).j_inv() ==␣
↪QuasiMeromorphicModularFormsRing(n=7)(j_inv)
True

sage: from sage.modular.modform_hecketriangle.space import WeakModularForms,␣
↪CuspForms
sage: MF = WeakModularForms(n=5, k=0)
sage: j_inv = MF.j_inv()
sage: j_inv in MF
True
sage: WeakModularFormsRing(n=5, red_hom=True).j_inv() == j_inv
True
sage: CuspForms(n=5, k=12).j_inv() == j_inv
True
sage: MF.disp_prec(3)
sage: j_inv
q^-1 + 79/(200*d) + 42877/(640000*d^2)*q + 12957/(2000000*d^3)*q^2 + O(q^3)

sage: WeakModularForms(n=infinity).j_inv()
q^-1 + 24 + 276*q + 2048*q^2 + 11202*q^3 + 49152*q^4 + O(q^5)

sage: WeakModularForms().j_inv()
q^-1 + 744 + 196884*q + 21493760*q^2 + 864299970*q^3 + 20245856256*q^4 + O(q^
↪5)
```

pol_ring ( )

Return the underlying polynomial ring used by `self`.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.graded_ring import␣
↪ModularFormsRing
sage: ModularFormsRing().pol_ring()
Multivariate Polynomial Ring in x, y, z, d over Integer Ring

sage: from sage.modular.modform_hecketriangle.space import CuspForms
sage: CuspForms(k=12, base_ring=AA).pol_ring()
Multivariate Polynomial Ring in x, y, z, d over Algebraic Real Field
```

rat_field ( )

Return the underlying rational field used by `self` to construct/represent elements.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.graded_ring import␣
↪ModularFormsRing
sage: ModularFormsRing().rat_field()
Fraction Field of Multivariate Polynomial Ring in x, y, z, d over Integer Ring
```

```
sage: from sage.modular.modform_hecketriangle.space import CuspForms
sage: CuspForms(k=12, base_ring=AA).rat_field()
Fraction Field of Multivariate Polynomial Ring in x, y, z, d over Algebraic␣
↪Real Field
```

reduce_type (*analytic_type=None*, *degree=None*)

Return a new space with analytic properties shared by both `self` and `analytic_type`, possibly reduced to its space of homogeneous elements of the given `degree` (if `degree` is set). Elements of the new space are contained in `self`.

INPUT:

- `analytic_type` – An `AnalyticType` or something which coerces into it (default: `None`).
- **`degree` – `None` (default) or the degree of the homogeneous component to which** `self` should be reduced.

OUTPUT:

The new reduced space.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.graded_ring import␣
↪QuasiModularFormsRing
sage: from sage.modular.modform_hecketriangle.space import QuasiModularForms

sage: MR = QuasiModularFormsRing()
sage: MR.reduce_type(["quasi", "cusp"])
QuasiCuspFormsRing(n=3) over Integer Ring

sage: MR.reduce_type("cusp", degree=(12,1))
CuspForms(n=3, k=12, ep=1) over Integer Ring

sage: MF=QuasiModularForms(k=6)
sage: MF.reduce_type("holo")
ModularForms(n=3, k=6, ep=-1) over Integer Ring

sage: MF.reduce_type([])
ZeroForms(n=3, k=6, ep=-1) over Integer Ring
```

CHAPTER

THREE

MODULAR FORMS FOR HECKE TRIANGLE GROUPS

AUTHORS:

- Jonas Jermann (2013): initial version

class `sage.modular.modform_hecketriangle.abstract_space.`**`FormsSpace_abstract`**(*group*, *base_ring*, *k*, *ep*, *n*)

Bases: *`sage.modular.modform_hecketriangle.abstract_ring.FormsRing_abstract`*

Abstract (Hecke) forms space.

This should never be called directly. Instead one should instantiate one of the derived classes of this class.

`Element`

alias of *`FormsElement`*

`F_basis`(*m*, *order_1=0*)

Returns a weakly holomorphic element of `self` (extended if necessarily) determined by the property that the Fourier expansion is of the form is of the form `q^m + O(q^(order_inf + 1))` , where `order_inf = self._l1 -order_1` .

In particular for all `m <= order_inf` these elements form a basis of the space of weakly holomorphic modular forms of the corresponding degree in case `n!=infinity` .

If `n=infinity` a non-trivial order of `-1` can be specified through the parameter `order_1` (default: 0). Otherwise it is ignored.

INPUT:

- `m` – An integer `m <= self._l1` .
- **`order_1` – The order at `-1` of `F_simple` (default: 0).** This parameter is ignored if `n != infinity` .

OUTPUT:

The corresponding element in (possibly an extension of) `self` . Note that the order at `-1` of the resulting element may be bigger than `order_1` (rare).

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.space import WeakModularForms,␣
↪CuspForms
sage: MF = WeakModularForms(n=5, k=62/3, ep=-1)
sage: MF.disp_prec(MF._l1+2)
sage: MF.weight_parameters()
(2, 3)
```

```
sage: MF.F_basis(2)
q^2 - 41/(200*d)*q^3 + O(q^4)
sage: MF.F_basis(1)
q - 13071/(640000*d^2)*q^3 + O(q^4)
sage: MF.F_basis(0)
1 - 277043/(192000000*d^3)*q^3 + O(q^4)
sage: MF.F_basis(-2)
q^-2 - 162727620113/(40960000000000000*d^5)*q^3 + O(q^4)
sage: MF.F_basis(-2).parent() == MF
True

sage: MF = CuspForms(n=4, k=-2, ep=1)
sage: MF.weight_parameters()
(-1, 3)

sage: MF.F_basis(-1).parent()
WeakModularForms(n=4, k=-2, ep=1) over Integer Ring
sage: MF.F_basis(-1).parent().disp_prec(MF._l1+2)
sage: MF.F_basis(-1)
q^-1 + 80 + O(q)
sage: MF.F_basis(-2)
q^-2 + 400 + O(q)

sage: MF = WeakModularForms(n=infinity, k=14, ep=-1)
sage: MF.F_basis(3)
q^3 - 48*q^4 + O(q^5)
sage: MF.F_basis(2)
q^2 - 1152*q^4 + O(q^5)
sage: MF.F_basis(1)
q - 18496*q^4 + O(q^5)
sage: MF.F_basis(0)
1 - 224280*q^4 + O(q^5)
sage: MF.F_basis(-1)
q^-1 - 2198304*q^4 + O(q^5)

sage: MF.F_basis(3, order_1=-1)
q^3 + O(q^5)
sage: MF.F_basis(1, order_1=2)
q - 300*q^3 - 4096*q^4 + O(q^5)
sage: MF.F_basis(0, order_1=2)
1 - 24*q^2 - 2048*q^3 - 98328*q^4 + O(q^5)
sage: MF.F_basis(-1, order_1=2)
q^-1 - 18150*q^3 - 1327104*q^4 + O(q^5)
```

`F_basis_pol` (*m*, *order_1=0*)

Returns a polynomial corresponding to the basis element of the correponding space of weakly holomorphic forms of the same degree as `self`. The basis element is determined by the property that the Fourier expansion is of the form `q^m + O(q^(order_inf + 1))`, where `order_inf = self._l1 -order_1`.

If `n=infinity` a non-trivial order of `-1` can be specified through the parameter `order_1` (default: 0). Otherwise it is ignored.

INPUT:

- `m` – An integer `m <= self._l1`.
- **`order_1` – The order at `-1` of `F_simple` (default: 0).** This parameter is ignored if `n !=`

    infinity.

OUTPUT:

A polynomial in x,y,z,d , corresponding to f_rho,f_i,E2 and the (possibly) transcendental parameter d .

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.space import WeakModularForms
sage: MF = WeakModularForms(n=5, k=62/3, ep=-1)
sage: MF.weight_parameters()
(2, 3)

sage: MF.F_basis_pol(2)
x^13*y*d^2 - 2*x^8*y^3*d^2 + x^3*y^5*d^2
sage: MF.F_basis_pol(1)
(-81*x^13*y*d + 62*x^8*y^3*d + 19*x^3*y^5*d)/(-100)
sage: MF.F_basis_pol(0)
(141913*x^13*y + 168974*x^8*y^3 + 9113*x^3*y^5)/320000

sage: MF(MF.F_basis_pol(2)).q_expansion(prec=MF._l1+2)
q^2 - 41/(200*d)*q^3 + O(q^4)
sage: MF(MF.F_basis_pol(1)).q_expansion(prec=MF._l1+1)
q + O(q^3)
sage: MF(MF.F_basis_pol(0)).q_expansion(prec=MF._l1+1)
1 + O(q^3)
sage: MF(MF.F_basis_pol(-2)).q_expansion(prec=MF._l1+1)
q^-2 + O(q^3)
sage: MF(MF.F_basis_pol(-2)).parent()
WeakModularForms(n=5, k=62/3, ep=-1) over Integer Ring

sage: MF = WeakModularForms(n=4, k=-2, ep=1)
sage: MF.weight_parameters()
(-1, 3)

sage: MF.F_basis_pol(-1)
x^3/(x^4*d - y^2*d)
sage: MF.F_basis_pol(-2)
(9*x^7 + 23*x^3*y^2)/(32*x^8*d^2 - 64*x^4*y^2*d^2 + 32*y^4*d^2)

sage: MF(MF.F_basis_pol(-1)).q_expansion(prec=MF._l1+2)
q^-1 + 5/(16*d) + O(q)
sage: MF(MF.F_basis_pol(-2)).q_expansion(prec=MF._l1+2)
q^-2 + 25/(4096*d^2) + O(q)

sage: MF = WeakModularForms(n=infinity, k=14, ep=-1)
sage: MF.F_basis_pol(3)
-y^7*d^3 + 3*x*y^5*d^3 - 3*x^2*y^3*d^3 + x^3*y*d^3
sage: MF.F_basis_pol(2)
(3*y^7*d^2 - 17*x*y^5*d^2 + 25*x^2*y^3*d^2 - 11*x^3*y*d^2)/(-8)
sage: MF.F_basis_pol(1)
(-75*y^7*d + 225*x*y^5*d - 1249*x^2*y^3*d + 1099*x^3*y*d)/1024
sage: MF.F_basis_pol(0)
(41*y^7 - 147*x*y^5 - 1365*x^2*y^3 - 2625*x^3*y)/(-4096)
sage: MF.F_basis_pol(-1)
(-9075*y^9 + 36300*x*y^7 - 718002*x^2*y^5 - 4928052*x^3*y^3 - 2769779*x^4*y)/
↪(8388608*y^2*d - 8388608*x*d)

sage: MF.F_basis_pol(3, order_1=-1)
```

```
(-3*y^9*d^3 + 16*x*y^7*d^3 - 30*x^2*y^5*d^3 + 24*x^3*y^3*d^3 - 7*x^4*y*d^3)/(-
↪4*x)
sage: MF.F_basis_pol(1, order_1=2)
-x^2*y^3*d + x^3*y*d
sage: MF.F_basis_pol(0, order_1=2)
(-3*x^2*y^3 - 5*x^3*y)/(-8)
sage: MF.F_basis_pol(-1, order_1=2)
(-81*x^2*y^5 - 606*x^3*y^3 - 337*x^4*y)/(1024*y^2*d - 1024*x*d)
```

F_simple (*order_1=0*)

Return a (the most) simple normalized element of `self` corresponding to the weight parameters `l1=self._l1` and `l2=self._l2`. If the element does not lie in `self` the type of its parent is extended accordingly.

The main part of the element is given by the `(l1 -order_1)` -th power of `f_inf`, up to a small holomorphic correction factor.

INPUT:

- **`order_1` – An integer (default: 0) denoting the desired order at** `-1` in the case `n = infinity`. If `n != infinity` the parameter is ignored.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.space import WeakModularForms
sage: MF = WeakModularForms(n=18, k=-7, ep=-1)
sage: MF.disp_prec(1)
sage: MF.F_simple()
q^-3 + 16/(81*d)*q^-2 - 4775/(104976*d^2)*q^-1 - 14300/(531441*d^3) + O(q)
sage: MF.F_simple() == MF.f_inf()^MF._l1 * MF.f_rho()^MF._l2 * MF.f_i()
True

sage: from sage.modular.modform_hecketriangle.space import CuspForms,␣
↪ModularForms
sage: MF = CuspForms(n=5, k=2, ep=-1)
sage: MF._l1
-1
sage: MF.F_simple().parent()
WeakModularForms(n=5, k=2, ep=-1) over Integer Ring

sage: MF = ModularForms(n=infinity, k=8, ep=1)
sage: MF.F_simple().reduced_parent()
ModularForms(n=+Infinity, k=8, ep=1) over Integer Ring
sage: MF.F_simple()
q^2 - 16*q^3 + 120*q^4 + O(q^5)
sage: MF.F_simple(order_1=2)
1 + 32*q + 480*q^2 + 4480*q^3 + 29152*q^4 + O(q^5)
```

Faber_pol (*m*, *order_1=0*, *fix_d=False*, *d_num_prec=None*)

Return the `m` ‘th Faber polynomial of `self`.

Namely a polynomial `P(q)` such that `P(J_inv)*F_simple(order_1)` has a Fourier expansion of the form `q^m + O(q^(order_inf + 1))`. where `order_inf = self._l1 -order_1` and `d^(order_inf -m)*P(q)` is a monic polynomial of degree `order_inf -m`.

If `n=infinity` a non-trivial order of `-1` can be specified through the parameter `order_1` (default: 0). Otherwise it is ignored.

The Faber polynomials are e.g. used to construct a basis of weakly holomorphic forms and to recover such forms from their initial Fourier coefficients.

INPUT:

- m – An integer m <= order_inf = self._l1 -order_1 .
- **order_1 – The order at -1 of F_simple (default: 0).** This parameter is ignored if n != infinity .
- **fix_d – If False (default) a formal parameter is used for d .** If True then the numerical value of d is used (resp. an exact value if the group is arithmetic). Otherwise the given value is used for d .
- **d_num_prec – The precision to be used if a numerical value for d is substituted.** Default: None in which case the default numerical precision of self.parent() is used.

OUTPUT:

The corresponding Faber polynomial P(q) .

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.space import WeakModularForms
sage: MF = WeakModularForms(n=5, k=62/3, ep=-1)
sage: MF.weight_parameters()
(2, 3)

sage: MF.Faber_pol(2)
1
sage: MF.Faber_pol(1)
1/d*q - 19/(100*d)
sage: MF.Faber_pol(0)
1/d^2*q^2 - 117/(200*d^2)*q + 9113/(320000*d^2)
sage: MF.Faber_pol(-2)
1/d^4*q^4 - 11/(8*d^4)*q^3 + 41013/(80000*d^4)*q^2 - 2251291/(48000000*d^4)*q␣
↪+ 1974089431/(4915200000000*d^4)
sage: (MF.Faber_pol(2)(MF.J_inv())*MF.F_simple()).q_expansion(prec=MF._l1+2)
q^2 - 41/(200*d)*q^3 + O(q^4)
sage: (MF.Faber_pol(1)(MF.J_inv())*MF.F_simple()).q_expansion(prec=MF._l1+1)
q + O(q^3)
sage: (MF.Faber_pol(0)(MF.J_inv())*MF.F_simple()).q_expansion(prec=MF._l1+1)
1 + O(q^3)
sage: (MF.Faber_pol(-2)(MF.J_inv())*MF.F_simple()).q_expansion(prec=MF._l1+1)
q^-2 + O(q^3)

sage: MF.Faber_pol(2, fix_d=1)
1
sage: MF.Faber_pol(1, fix_d=1)
q - 19/100
sage: MF.Faber_pol(-2, fix_d=1)
q^4 - 11/8*q^3 + 41013/80000*q^2 - 2251291/48000000*q + 1974089431/
↪4915200000000
sage: (MF.Faber_pol(2, fix_d=1)(MF.J_inv())*MF.F_simple()).q_
↪expansion(prec=MF._l1+2, fix_d=1)
q^2 - 41/200*q^3 + O(q^4)
sage: (MF.Faber_pol(-2)(MF.J_inv())*MF.F_simple()).q_expansion(prec=MF._l1+1,␣
↪fix_d=1)
q^-2 + O(q^3)

sage: MF = WeakModularForms(n=4, k=-2, ep=1)
sage: MF.weight_parameters()
(-1, 3)
```

```
sage: MF.Faber_pol(-1)
1
sage: MF.Faber_pol(-2, fix_d=True)
256*q - 184
sage: MF.Faber_pol(-3, fix_d=True)
65536*q^2 - 73728*q + 14364
sage: (MF.Faber_pol(-1, fix_d=True)(MF.J_inv())*MF.F_simple()).q_
↪expansion(prec=MF._l1+2, fix_d=True)
q^-1 + 80 + O(q)
sage: (MF.Faber_pol(-2, fix_d=True)(MF.J_inv())*MF.F_simple()).q_
↪expansion(prec=MF._l1+2, fix_d=True)
q^-2 + 400 + O(q)
sage: (MF.Faber_pol(-3)(MF.J_inv())*MF.F_simple()).q_expansion(prec=MF._l1+2,␣
↪fix_d=True)
q^-3 + 2240 + O(q)

sage: MF = WeakModularForms(n=infinity, k=14, ep=-1)
sage: MF.Faber_pol(3)
1
sage: MF.Faber_pol(2)
1/d*q + 3/(8*d)
sage: MF.Faber_pol(1)
1/d^2*q^2 + 75/(1024*d^2)
sage: MF.Faber_pol(0)
1/d^3*q^3 - 3/(8*d^3)*q^2 + 3/(512*d^3)*q + 41/(4096*d^3)
sage: MF.Faber_pol(-1)
1/d^4*q^4 - 3/(4*d^4)*q^3 + 81/(1024*d^4)*q^2 + 9075/(8388608*d^4)
sage: (MF.Faber_pol(-1)(MF.J_inv())*MF.F_simple()).q_expansion(prec=MF._l1 +␣
↪1)
q^-1 + O(q^4)

sage: MF.Faber_pol(3, order_1=-1)
1/d*q + 3/(4*d)
sage: MF.Faber_pol(1, order_1=2)
1
sage: MF.Faber_pol(0, order_1=2)
1/d*q - 3/(8*d)
sage: MF.Faber_pol(-1, order_1=2)
1/d^2*q^2 - 3/(4*d^2)*q + 81/(1024*d^2)
sage: (MF.Faber_pol(-1, order_1=2)(MF.J_inv())*MF.F_simple(order_1=2)).q_
↪expansion(prec=MF._l1 + 1)
q^-1 - 9075/(8388608*d^4)*q^3 + O(q^4)
```

FormsElement

alias of *FormsElement*

ambient_coordinate_vector (*v*)

Return the coordinate vector of the element `v` in `self.module()` with respect to the basis from `self.ambient_space`.

NOTE:

Elements use this method (from their parent) to calculate their coordinates.

INPUT:

- `v` – An element of `self`.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.space import ModularForms
sage: MF = ModularForms(n=4, k=24, ep=-1)
sage: MF.ambient_coordinate_vector(MF.gen(0)).parent()
Vector space of dimension 3 over Fraction Field of Univariate Polynomial Ring␣
↪in d over Integer Ring
sage: MF.ambient_coordinate_vector(MF.gen(0))
(1, 0, 0)
sage: subspace = MF.subspace([MF.gen(0), MF.gen(2)])
sage: subspace.ambient_coordinate_vector(subspace.gen(0)).parent()
Vector space of degree 3 and dimension 2 over Fraction Field of Univariate␣
↪Polynomial Ring in d over Integer Ring
Basis matrix:
[1 0 0]
[0 0 1]
sage: subspace.ambient_coordinate_vector(subspace.gen(0))
(1, 0, 0)
```

ambient_module ( )

Return the module associated to the ambient space of self.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.space import ModularForms
sage: MF = ModularForms(k=12)
sage: MF.ambient_module()
Vector space of dimension 2 over Fraction Field of Univariate Polynomial Ring␣
↪in d over Integer Ring
sage: MF.ambient_module() == MF.module()
True
sage: subspace = MF.subspace([MF.gen(0)])
sage: subspace.ambient_module() == MF.module()
True
```

ambient_space ( )

Return the ambient space of self.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.space import ModularForms
sage: MF = ModularForms(k=12)
sage: MF.ambient_space()
ModularForms(n=3, k=12, ep=1) over Integer Ring
sage: MF.ambient_space() == MF
True
sage: subspace = MF.subspace([MF.gen(0)])
sage: subspace
Subspace of dimension 1 of ModularForms(n=3, k=12, ep=1) over Integer Ring
sage: subspace.ambient_space() == MF
True
```

aut_factor (*gamma*, *t*)

The automorphy factor of `self`.

INPUT:

- `gamma` – An element of the group of `self`.
- `t` – An element of the upper half plane.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.space import ModularForms
sage: MF = ModularForms(n=8, k=4, ep=1)
sage: full_factor = lambda mat, t: (mat[1][0]*t+mat[1][1])**4
sage: T = MF.group().T()
sage: S = MF.group().S()
sage: i = AlgebraicField()(i)
sage: z = 1 + i/2

sage: MF.aut_factor(S, z)
3/2*I - 7/16
sage: MF.aut_factor(-T^(-2), z)
1
sage: MF.aut_factor(MF.group().V(6), z)
173.2640595631...? + 343.8133289126...?*I
sage: MF.aut_factor(S, z) == full_factor(S, z)
True
sage: MF.aut_factor(T, z) == full_factor(T, z)
True
sage: MF.aut_factor(MF.group().V(6), z) == full_factor(MF.group().V(6), z)
True

sage: MF = ModularForms(n=7, k=14/5, ep=-1)
sage: T = MF.group().T()
sage: S = MF.group().S()

sage: MF.aut_factor(S, z)
1.3655215324256...? + 0.056805991182877...?*I
sage: MF.aut_factor(-T^(-2), z)
1
sage: MF.aut_factor(S, z) == MF.ep() * (z/i)^MF.weight()
True
sage: MF.aut_factor(MF.group().V(6), z)
13.23058830577...? + 15.71786610686...?*I
```

change_ring (*new_base_ring*)

Return the same space as `self` but over a new base ring `new_base_ring`.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.space import CuspForms
sage: CuspForms(n=5, k=24).change_ring(CC)
CuspForms(n=5, k=24, ep=1) over Complex Field with 53 bits of precision
```

construct_form (*laurent_series*, *order_1=0*, *check=True*, *rationalize=False*)

Tries to construct an element of self with the given Fourier expansion. The assumption is made that the specified Fourier expansion corresponds to a weakly holomorphic modular form.

If the precision is too low to determine the element an exception is raised.

INPUT:

- `laurent_series` – A Laurent or Power series.
- **`order_1` – A lower bound for the order at `-1` of the form (default: 0).** If `n!=infinity` this parameter is ignored.
- **`check` – If `True` (default) then the series expansion of the constructed** form is compared against the given series.

- **`rationalize` – If `True` (default: `False`) then the series is** *rationalized* beforehand. Note that in non-exact or non-arithmetic cases this is experimental and extremely unreliable!

OUTPUT:

If possible: An element of self with the same initial Fourier expansion as `laurent_series`.

Note: For modular spaces it is also possible to call `self(laurent_series)` instead.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.space import CuspForms
sage: Delta = CuspForms(k=12).Delta()
sage: qexp = Delta.q_expansion(prec=2)
sage: qexp.parent()
Power Series Ring in q over Fraction Field of Univariate Polynomial Ring in d␣
↪over Integer Ring
sage: qexp
q + O(q^2)
sage: CuspForms(k=12).construct_form(qexp) == Delta
True

sage: from sage.modular.modform_hecketriangle.space import WeakModularForms
sage: J_inv = WeakModularForms(n=7).J_inv()
sage: qexp2 = J_inv.q_expansion(prec=1)
sage: qexp2.parent()
Laurent Series Ring in q over Fraction Field of Univariate Polynomial Ring in␣
↪d over Integer Ring
sage: qexp2
d*q^-1 + 151/392 + O(q)
sage: WeakModularForms(n=7).construct_form(qexp2) == J_inv
True

sage: MF = WeakModularForms(n=5, k=62/3, ep=-1)
sage: MF.default_prec(MF._l1+1)
sage: d = MF.get_d()
sage: MF.weight_parameters()
(2, 3)
sage: el2 = d*MF.F_basis(2) + 2*MF.F_basis(1) + MF.F_basis(-2)
sage: qexp2 = el2.q_expansion()
sage: qexp2.parent()
Laurent Series Ring in q over Fraction Field of Univariate Polynomial Ring in␣
↪d over Integer Ring
sage: qexp2
q^-2 + 2*q + d*q^2 + O(q^3)
sage: WeakModularForms(n=5, k=62/3, ep=-1).construct_form(qexp2) == el2
True

sage: MF = WeakModularForms(n=infinity, k=-2, ep=-1)
sage: el3 = MF.f_i()/MF.f_inf() + MF.f_i()*MF.f_inf()/MF.E4()^2
sage: MF.quasi_part_dimension(min_exp=-1, order_1=-2)
3
sage: prec = MF._l1 + 3
sage: qexp3 = el3.q_expansion(prec)
sage: qexp3
q^-1 - 1/(4*d) + ((1024*d^2 - 33)/(1024*d^2))*q + O(q^2)
sage: MF.construct_form(qexp3, order_1=-2) == el3
True
sage: MF.construct_form(el3.q_expansion(prec + 1), order_1=-3) == el3
True
```

```
sage: WF = WeakModularForms(n=14)
sage: qexp = WF.J_inv().q_expansion_fixed_d(d_num_prec=1000)
sage: qexp.parent()
Laurent Series Ring in q over Real Field with 1000 bits of precision
sage: WF.construct_form(qexp, rationalize=True) == WF.J_inv()
doctest:...: UserWarning: Using an experimental rationalization of␣
↪coefficients, please check the result for correctness!
True
```

construct_quasi_form (*laurent_series*, *order_1=0*, *check=True*, *rationalize=False*)

Try to construct an element of self with the given Fourier expansion. The assumption is made that the specified Fourier expansion corresponds to a weakly holomorphic quasi modular form.

If the precision is too low to determine the element an exception is raised.

INPUT:

- `laurent_series` – A Laurent or Power series.
- **`order_1` – A lower bound for the order at `-1` for all quasi parts of the** form (default: 0). If `n!=infinity` this parameter is ignored.
- **`check` – If `True` (default) then the series expansion of the constructed** form is compared against the given (rationalized) series.
- **`rationalize` – If `True` (default: `False`) then the series is** *rationalized* beforehand. Note that in non-exact or non-arithmetic cases this is experimental and extremely unreliable!

OUTPUT:

If possible: An element of self with the same initial Fourier expansion as `laurent_series`.

Note: For non modular spaces it is also possible to call `self(laurent_series)` instead. Also note that this function works much faster if a corresponding (cached) `q_basis` is available.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.space import␣
↪QuasiWeakModularForms, ModularForms, QuasiModularForms, QuasiCuspForms
sage: QF = QuasiWeakModularForms(n=8, k=10/3, ep=-1)
sage: el = QF.quasi_part_gens(min_exp=-1)[4]
sage: prec = QF.required_laurent_prec(min_exp=-1)
sage: prec
5
sage: qexp = el.q_expansion(prec=prec)
sage: qexp
q^-1 - 19/(64*d) - 7497/(262144*d^2)*q + 15889/(8388608*d^3)*q^2 + 543834047/
↪(1649267441664*d^4)*q^3 + 711869853/(43980465111040*d^5)*q^4 + O(q^5)
sage: qexp.parent()
Laurent Series Ring in q over Fraction Field of Univariate Polynomial Ring in␣
↪d over Integer Ring
sage: constructed_el = QF.construct_quasi_form(qexp)
sage: constructed_el.parent()
QuasiWeakModularForms(n=8, k=10/3, ep=-1) over Integer Ring
sage: el == constructed_el
True

If a q_basis is available the construction uses a different algorithm which␣
↪we also check::
```

```
sage: basis = QF.q_basis(min_exp=-1)
sage: QF(qexp) == constructed_el
True

sage: MF = ModularForms(k=36)
sage: el2 = MF.quasi_part_gens(min_exp=2)[1]
sage: prec = MF.required_laurent_prec(min_exp=2)
sage: prec
4
sage: qexp2 = el2.q_expansion(prec=prec + 1)
sage: qexp2
q^3 - 1/(24*d)*q^4 + O(q^5)
sage: qexp2.parent()
Power Series Ring in q over Fraction Field of Univariate Polynomial Ring in d␣
↪over Integer Ring
sage: constructed_el2 = MF.construct_quasi_form(qexp2)
sage: constructed_el2.parent()
ModularForms(n=3, k=36, ep=1) over Integer Ring
sage: el2 == constructed_el2
True

sage: QF = QuasiModularForms(k=2)
sage: q = QF.get_q()
sage: qexp3 = 1 + O(q)
sage: QF(qexp3)
1 - 24*q - 72*q^2 - 96*q^3 - 168*q^4 + O(q^5)
sage: QF(qexp3) == QF.E2()
True

sage: QF = QuasiWeakModularForms(n=infinity, k=2, ep=-1)
sage: el4 = QF.f_i() + QF.f_i()^3/QF.E4()
sage: prec = QF.required_laurent_prec(order_1=-1)
sage: qexp4 = el4.q_expansion(prec=prec)
sage: qexp4
2 - 7/(4*d)*q + 195/(256*d^2)*q^2 - 903/(4096*d^3)*q^3 + 41987/(1048576*d^
↪4)*q^4 - 181269/(33554432*d^5)*q^5 + O(q^6)
sage: QF.construct_quasi_form(qexp4, check=False) == el4
False
sage: QF.construct_quasi_form(qexp4, order_1=-1) == el4
True

sage: QF = QuasiCuspForms(n=8, k=22/3, ep=-1)
sage: el = QF(QF.f_inf()*QF.E2())
sage: qexp = el.q_expansion_fixed_d(d_num_prec=1000)
sage: qexp.parent()
Power Series Ring in q over Real Field with 1000 bits of precision
sage: QF.construct_quasi_form(qexp, rationalize=True) == el
True
```

construction ( )

Return a functor that constructs `self` (used by the coercion machinery).

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.space import QuasiModularForms
sage: QuasiModularForms(n=4, k=2, ep=1, base_ring=CC).construction()
(QuasiModularFormsFunctor(n=4, k=2, ep=1),
 BaseFacade(Complex Field with 53 bits of precision))
```

```
sage: from sage.modular.modform_hecketriangle.space import ModularForms
sage: MF=ModularForms(k=12)
sage: MF.subspace([MF.gen(1)]).construction()
(FormsSubSpaceFunctor with 1 generator for the ModularFormsFunctor(n=3, k=12,␣
↪ep=1), BaseFacade(Integer Ring))
```

contains_coeff_ring ( )

Return whether `self` contains its coefficient ring.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.space import QuasiModularForms
sage: QuasiModularForms(k=0, ep=1, n=8).contains_coeff_ring()
True
sage: QuasiModularForms(k=0, ep=-1, n=8).contains_coeff_ring()
False
```

coordinate_vector (*v*)

This method should be overloaded by subclasses.

Return the coordinate vector of the element `v` with respect to `self.gens()` .

NOTE:

Elements use this method (from their parent) to calculate their coordinates.

INPUT:

•`v` – An element of `self` .

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.space import ModularForms
sage: MF = ModularForms(n=4, k=24, ep=-1)
sage: MF.coordinate_vector(MF.gen(0)).parent() # defined in space.py
Vector space of dimension 3 over Fraction Field of Univariate Polynomial Ring␣
↪in d over Integer Ring
sage: MF.coordinate_vector(MF.gen(0))          # defined in space.py
(1, 0, 0)
sage: subspace = MF.subspace([MF.gen(0), MF.gen(2)])
sage: subspace.coordinate_vector(subspace.gen(0)).parent()  # defined in␣
↪subspace.py
Vector space of dimension 2 over Fraction Field of Univariate Polynomial Ring␣
↪in d over Integer Ring
sage: subspace.coordinate_vector(subspace.gen(0))           # defined in␣
↪subspace.py
(1, 0)
```

degree ( )

Return the degree of `self` .

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.space import ModularForms
sage: MF = ModularForms(n=4, k=24, ep=-1)
sage: MF.degree()
3
sage: MF.subspace([MF.gen(0), MF.gen(2)]).degree() # defined in subspace.py
3
```

dimension ( )

Return the dimension of `self` .

This method should be overloaded by subclasses.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.space import␣
↪QuasiMeromorphicModularForms
sage: QuasiMeromorphicModularForms(k=2, ep=-1).dimension()
+Infinity
```

element_from_ambient_coordinates (*vec*)

If `self` has an associated free module, then return the element of `self` corresponding to the given `vec` . Otherwise raise an exception.

INPUT:

- `vec` – An element of `self.module()` or `self.ambient_module()` .

OUTPUT:

An element of `self` corresponding to `vec` .

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.space import ModularForms
sage: MF = ModularForms(k=24)
sage: MF.dimension()
3
sage: el = MF.element_from_ambient_coordinates([1,1,1])
sage: el == MF.element_from_coordinates([1,1,1])
True
sage: el.parent() == MF
True

sage: subspace = MF.subspace([MF.gen(0), MF.gen(1)])
sage: el = subspace.element_from_ambient_coordinates([1,1,0])
sage: el
1 + q + 52611660*q^3 + 39019412128*q^4 + O(q^5)
sage: el.parent() == subspace
True
```

element_from_coordinates (*vec*)

If `self` has an associated free module, then return the element of `self` corresponding to the given coordinate vector `vec` . Otherwise raise an exception.

INPUT:

- `vec` – A coordinate vector with respect to `self.gens()` .

OUTPUT:

An element of `self` corresponding to the coordinate vector `vec` .

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.space import ModularForms
sage: MF = ModularForms(k=24)
sage: MF.dimension()
3
sage: el = MF.element_from_coordinates([1,1,1])
```

```
sage: el
1 + q + q^2 + 52611612*q^3 + 39019413208*q^4 + O(q^5)
sage: el == MF.gen(0) + MF.gen(1) + MF.gen(2)
True
sage: el.parent() == MF
True

sage: subspace = MF.subspace([MF.gen(0), MF.gen(1)])
sage: el = subspace.element_from_coordinates([1,1])
sage: el
1 + q + 52611660*q^3 + 39019412128*q^4 + O(q^5)
sage: el == subspace.gen(0) + subspace.gen(1)
True
sage: el.parent() == subspace
True
```

ep ( )

Return the multiplier of (elements of) `self`.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.space import QuasiModularForms
sage: QuasiModularForms(n=16, k=16/7, ep=-1).ep()
-1
```

faber_pol (*m*, *order_1=0*, *fix_d=False*, *d_num_prec=None*)

If `n=infinity` a non-trivial order of `-1` can be specified through the parameter `order_1` (default: 0). Otherwise it is ignored. Return the m'th Faber polynomial of `self` with a different normalization based on `j_inv` instead of `J_inv`.

Namely a polynomial `p(q)` such that `p(j_inv)*F_simple()` has a Fourier expansion of the form `q^m + O(q^(order_inf + 1))`. where `order_inf = self._l1 -order_1` and `p(q)` is a monic polynomial of degree `order_inf -m`.

If `n=infinity` a non-trivial order of `-1` can be specified through the parameter `order_1` (default: 0). Otherwise it is ignored.

The relation to `Faber_pol` is: `faber_pol(q) = Faber_pol(d*q)`.

INPUT:

- `m` – An integer `m <= self._l1 -order_1`.
- **`order_1` – The order at `-1` of `F_simple` (default: 0).** This parameter is ignored if `n != infinity`.
- **`fix_d` – If `False` (default) a formal parameter is used for `d`.** If `True` then the numerical value of `d` is used (resp. an exact value if the group is arithmetic). Otherwise the given value is used for `d`.
- **`d_num_prec` – The precision to be used if a numerical value for `d` is substituted.** Default: `None` in which case the default numerical precision of `self.parent()` is used.

OUTPUT:

The corresponding Faber polynomial `p(q)`.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.space import WeakModularForms
sage: MF = WeakModularForms(n=5, k=62/3, ep=-1)
```

```
sage: MF.weight_parameters()
(2, 3)

sage: MF.faber_pol(2)
1
sage: MF.faber_pol(1)
q - 19/(100*d)
sage: MF.faber_pol(0)
q^2 - 117/(200*d)*q + 9113/(320000*d^2)
sage: MF.faber_pol(-2)
q^4 - 11/(8*d)*q^3 + 41013/(80000*d^2)*q^2 - 2251291/(48000000*d^3)*q +␣
↪1974089431/(4915200000000*d^4)
sage: (MF.faber_pol(2)(MF.j_inv())*MF.F_simple()).q_expansion(prec=MF._l1+2)
q^2 - 41/(200*d)*q^3 + O(q^4)
sage: (MF.faber_pol(1)(MF.j_inv())*MF.F_simple()).q_expansion(prec=MF._l1+1)
q + O(q^3)
sage: (MF.faber_pol(0)(MF.j_inv())*MF.F_simple()).q_expansion(prec=MF._l1+1)
1 + O(q^3)
sage: (MF.faber_pol(-2)(MF.j_inv())*MF.F_simple()).q_expansion(prec=MF._l1+1)
q^-2 + O(q^3)

sage: MF = WeakModularForms(n=4, k=-2, ep=1)
sage: MF.weight_parameters()
(-1, 3)

sage: MF.faber_pol(-1)
1
sage: MF.faber_pol(-2, fix_d=True)
q - 184
sage: MF.faber_pol(-3, fix_d=True)
q^2 - 288*q + 14364
sage: (MF.faber_pol(-1, fix_d=True)(MF.j_inv())*MF.F_simple()).q_
↪expansion(prec=MF._l1+2, fix_d=True)
q^-1 + 80 + O(q)
sage: (MF.faber_pol(-2, fix_d=True)(MF.j_inv())*MF.F_simple()).q_
↪expansion(prec=MF._l1+2, fix_d=True)
q^-2 + 400 + O(q)
sage: (MF.faber_pol(-3)(MF.j_inv())*MF.F_simple()).q_expansion(prec=MF._l1+2,␣
↪fix_d=True)
q^-3 + 2240 + O(q)

sage: MF = WeakModularForms(n=infinity, k=14, ep=-1)
sage: MF.faber_pol(3)
1
sage: MF.faber_pol(2)
q + 3/(8*d)
sage: MF.faber_pol(1)
q^2 + 75/(1024*d^2)
sage: MF.faber_pol(0)
q^3 - 3/(8*d)*q^2 + 3/(512*d^2)*q + 41/(4096*d^3)
sage: MF.faber_pol(-1)
q^4 - 3/(4*d)*q^3 + 81/(1024*d^2)*q^2 + 9075/(8388608*d^4)
sage: (MF.faber_pol(-1)(MF.j_inv())*MF.F_simple()).q_expansion(prec=MF._l1 +␣
↪1)
q^-1 + O(q^4)

sage: MF.faber_pol(3, order_1=-1)
q + 3/(4*d)
```

```
sage: MF.faber_pol(1, order_1=2)
1
sage: MF.faber_pol(0, order_1=2)
q - 3/(8*d)
sage: MF.faber_pol(-1, order_1=2)
q^2 - 3/(4*d)*q + 81/(1024*d^2)
sage: (MF.faber_pol(-1, order_1=2)(MF.j_inv())*MF.F_simple(order_1=2)).q_
↪expansion(prec=MF._l1 + 1)
q^-1 - 9075/(8388608*d^4)*q^3 + O(q^4)
```

gen (*k=0*)

Return the `k` 'th basis element of `self` if possible (default: `k=0`).

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.space import ModularForms
sage: ModularForms(k=12).gen(1).parent()
ModularForms(n=3, k=12, ep=1) over Integer Ring
sage: ModularForms(k=12).gen(1)
q - 24*q^2 + 252*q^3 - 1472*q^4 + O(q^5)
```

gens ( )

This method should be overloaded by subclasses.

Return a basis of `self` .

Note that the coordinate vector of elements of `self` are with respect to this basis.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.space import ModularForms
sage: ModularForms(k=12).gens() # defined in space.py
[1 + 196560*q^2 + 16773120*q^3 + 398034000*q^4 + O(q^5),
 q - 24*q^2 + 252*q^3 - 1472*q^4 + O(q^5)]
```

homogeneous_part (*k*, *ep*)

Since `self` already is a homogeneous component return `self` unless the degree differs in which case a `ValueError` is raised.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.space import␣
↪QuasiMeromorphicModularForms
sage: MF = QuasiMeromorphicModularForms(n=6, k=4)
sage: MF == MF.homogeneous_part(4,1)
True
sage: MF.homogeneous_part(5,1)
Traceback (most recent call last):
...
ValueError: QuasiMeromorphicModularForms(n=6, k=4, ep=1) over Integer Ring␣
↪already is homogeneous with degree (4, 1) != (5, 1)!
```

is_ambient ( )

Return whether `self` is an ambient space.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.space import ModularForms
sage: MF = ModularForms(k=12)
sage: MF.is_ambient()
```

```
True
sage: MF.subspace([MF.gen(0)]).is_ambient()
False
```

module ( )

Return the module associated to self.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.space import ModularForms
sage: MF = ModularForms(k=12)
sage: MF.module()
Vector space of dimension 2 over Fraction Field of Univariate Polynomial Ring␣
↪in d over Integer Ring
sage: subspace = MF.subspace([MF.gen(0)])
sage: subspace.module()
Vector space of degree 2 and dimension 1 over Fraction Field of Univariate␣
↪Polynomial Ring in d over Integer Ring
Basis matrix:
[1 0]
```

one ( )

Return the one element from the corresponding space of constant forms.

Note: The one element does not lie in `self` in general.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.space import CuspForms
sage: MF = CuspForms(k=12)
sage: MF.Delta()^0 == MF.one()
True
sage: (MF.Delta()^0).parent()
ModularForms(n=3, k=0, ep=1) over Integer Ring
```

one_element ( )

Return the one element from the corresponding space of constant forms.

Note: The one element does not lie in `self` in general.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.space import CuspForms
sage: MF = CuspForms(k=12)
sage: (MF.Delta()^(-1)).parent()
MeromorphicModularForms(n=3, k=-12, ep=1) over Integer Ring
sage: MF.one_element()
doctest:...: DeprecationWarning: .one_element() is deprecated. Use .one()␣
↪instead.
See http://trac.sagemath.org/17694 for details.
1 + O(q^5)
```

q_basis (*m=None*, *min_exp=0*, *order_1=0*)

Try to return a (basis) element of `self` with a Laurent series of the form `q^m + O(q^N)`, where `N=self.required_laurent_prec(min_exp)`.

If `m==None` the whole basis (with varying `m` 's) is returned if it exists.

INPUT:

- **`m` – An integer, indicating the desired initial Laurent exponent of the element.** If `m==None` (default) then the whole basis is returned.
- **`min_exp` – An integer, indicating the minimal Laurent exponent (for each quasi part)** of the subspace of `self` which should be considered (default: 0).
- **`order_1` – A lower bound for the order at `-1` of all quasi parts of the subspace** (default: 0). If `n!=infinity` this parameter is ignored.

OUTPUT:

The corresponding basis (if `m==None`) resp. the corresponding basis vector (if `m!=None`). If the basis resp. element doesn't exist an exception is raised.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.space import␣
↪QuasiWeakModularForms, ModularForms, QuasiModularForms
sage: QF = QuasiWeakModularForms(n=8, k=10/3, ep=-1)
sage: QF.default_prec(QF.required_laurent_prec(min_exp=-1))
sage: q_basis = QF.q_basis(min_exp=-1)
sage: q_basis
[q^-1 + O(q^5), 1 + O(q^5), q + O(q^5), q^2 + O(q^5), q^3 + O(q^5), q^4 + O(q^
↪5)]
sage: QF.q_basis(m=-1, min_exp=-1)
q^-1 + O(q^5)

sage: MF = ModularForms(k=36)
sage: MF.q_basis() == MF.gens()
True

sage: QF = QuasiModularForms(k=6)
sage: QF.required_laurent_prec()
3
sage: QF.q_basis()
[1 - 20160*q^3 - 158760*q^4 + O(q^5), q - 60*q^3 - 248*q^4 + O(q^5), q^2 +␣
↪8*q^3 + 30*q^4 + O(q^5)]

sage: QF = QuasiWeakModularForms(n=infinity, k=-2, ep=-1)
sage: QF.q_basis(order_1=-1)
[1 - 168*q^2 + 2304*q^3 - 19320*q^4 + O(q^5),
 q - 18*q^2 + 180*q^3 - 1316*q^4 + O(q^5)]
```

`quasi_part_dimension` (*r=None*, *min_exp=0*, *max_exp=+Infinity*, *order_1=0*)

Return the dimension of the subspace of `self` generated by `self.quasi_part_gens(r,min_exp,max_exp,order_1)` .

See *`quasi_part_gens()`* for more details.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.space import QuasiModularForms,␣
↪QuasiCuspForms, QuasiWeakModularForms
sage: MF = QuasiModularForms(n=5, k=6, ep=-1)
sage: [v.as_ring_element() for v in MF.gens()]
[f_rho^2*f_i, f_rho^3*E2, E2^3]
sage: MF.dimension()
3
sage: MF.quasi_part_dimension(r=0)
1
sage: MF.quasi_part_dimension(r=1)
```

```
1
sage: MF.quasi_part_dimension(r=2)
0
sage: MF.quasi_part_dimension(r=3)
1

sage: MF = QuasiCuspForms(n=5, k=18, ep=-1)
sage: MF.dimension()
8
sage: MF.quasi_part_dimension(r=0)
2
sage: MF.quasi_part_dimension(r=1)
2
sage: MF.quasi_part_dimension(r=2)
1
sage: MF.quasi_part_dimension(r=3)
1
sage: MF.quasi_part_dimension(r=4)
1
sage: MF.quasi_part_dimension(r=5)
1
sage: MF.quasi_part_dimension(min_exp=2, max_exp=2)
2

sage: MF = QuasiCuspForms(n=infinity, k=18, ep=-1)
sage: MF.quasi_part_dimension(r=1, min_exp=-2)
3
sage: MF.quasi_part_dimension()
12
sage: MF.quasi_part_dimension(order_1=3)
2

sage: MF = QuasiWeakModularForms(n=infinity, k=4, ep=1)
sage: MF.quasi_part_dimension(min_exp=2, order_1=-2)
4
sage: [v.order_at(-1) for v in MF.quasi_part_gens(r=0, min_exp=2, order_1=-2)]
[-2, -2]
```

quasi_part_gens (*r=None*, *min_exp=0*, *max_exp=+Infinity*, *order_1=0*)

Return a basis in `self` of the subspace of (quasi) weakly holomorphic forms which satisfy the specified properties on the quasi parts and the initial Fourier coefficient.

INPUT:

- **r – An integer or None (default), indicating** the desired power of `E2` If `r=None` then all possible powers (`r`) are choosen.
- **min_exp – An integer giving a lower bound for the** first non-trivial Fourier coefficient of the generators (default: 0).
- **max_exp – An integer or infinity (default) giving** an upper bound for the first non-trivial Fourier coefficient of the generators. If `max_exp==infinity` then no upper bound is assumed.
- **order_1 – A lower bound for the order at -1 of all** quasi parts of the basis elements (default: 0). If `n!=infinity` this parameter is ignored.

OUTPUT:

A basis in `self` of the subspace of forms which are modular after dividing by `E2^r` and which have a

Fourier expansion of the form `q^m + O(q^(m+1))` with `min_exp <= m <= max_exp` for each quasi part (and at least the specified order at `-1` in case `n=infinity`). Note that linear combinations of forms/quasi parts maybe have a higher order at infinity than `max_exp`.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.space import␣
↪QuasiWeakModularForms
sage: QF = QuasiWeakModularForms(n=8, k=10/3, ep=-1)
sage: QF.default_prec(1)
sage: QF.quasi_part_gens(min_exp=-1)
[q^-1 + O(q), 1 + O(q), q^-1 - 9/(128*d) + O(q), 1 + O(q), q^-1 - 19/(64*d) +␣
↪O(q), q^-1 + 1/(64*d) + O(q)]

sage: QF.quasi_part_gens(min_exp=-1, max_exp=-1)
[q^-1 + O(q), q^-1 - 9/(128*d) + O(q), q^-1 - 19/(64*d) + O(q), q^-1 + 1/
↪(64*d) + O(q)]
sage: QF.quasi_part_gens(min_exp=-2, r=1)
[q^-2 - 9/(128*d)*q^-1 - 261/(131072*d^2) + O(q), q^-1 - 9/(128*d) + O(q), 1␣
↪+ O(q)]

sage: from sage.modular.modform_hecketriangle.space import ModularForms
sage: MF = ModularForms(k=36)
sage: MF.quasi_part_gens(min_exp=2)
[q^2 + 194184*q^4 + O(q^5), q^3 - 72*q^4 + O(q^5)]

sage: from sage.modular.modform_hecketriangle.space import QuasiModularForms
sage: MF = QuasiModularForms(n=5, k=6, ep=-1)
sage: MF.default_prec(2)
sage: MF.dimension()
3
sage: MF.quasi_part_gens(r=0)
[1 - 37/(200*d)*q + O(q^2)]
sage: MF.quasi_part_gens(r=0)[0] == MF.E6()
True
sage: MF.quasi_part_gens(r=1)
[1 + 33/(200*d)*q + O(q^2)]
sage: MF.quasi_part_gens(r=1)[0] == MF.E2()*MF.E4()
True
sage: MF.quasi_part_gens(r=2)
[]
sage: MF.quasi_part_gens(r=3)
[1 - 27/(200*d)*q + O(q^2)]
sage: MF.quasi_part_gens(r=3)[0] == MF.E2()^3
True

sage: from sage.modular.modform_hecketriangle.space import QuasiCuspForms,␣
↪CuspForms
sage: MF = QuasiCuspForms(n=5, k=18, ep=-1)
sage: MF.default_prec(4)
sage: MF.dimension()
8
sage: MF.quasi_part_gens(r=0)
[q - 34743/(640000*d^2)*q^3 + O(q^4), q^2 - 69/(200*d)*q^3 + O(q^4)]
sage: MF.quasi_part_gens(r=1)
[q - 9/(200*d)*q^2 + 37633/(640000*d^2)*q^3 + O(q^4),
 q^2 + 1/(200*d)*q^3 + O(q^4)]
sage: MF.quasi_part_gens(r=2)
[q - 1/(4*d)*q^2 - 24903/(640000*d^2)*q^3 + O(q^4)]
```

```
sage: MF.quasi_part_gens(r=3)
[q + 1/(10*d)*q^2 - 7263/(640000*d^2)*q^3 + O(q^4)]
sage: MF.quasi_part_gens(r=4)
[q - 11/(20*d)*q^2 + 53577/(640000*d^2)*q^3 + O(q^4)]
sage: MF.quasi_part_gens(r=5)
[q - 1/(5*d)*q^2 + 4017/(640000*d^2)*q^3 + O(q^4)]

sage: MF.quasi_part_gens(r=1)[0] == MF.E2() * CuspForms(n=5, k=16, ep=1).
↪gen(0)
True
sage: MF.quasi_part_gens(r=1)[1] == MF.E2() * CuspForms(n=5, k=16, ep=1).
↪gen(1)
True
sage: MF.quasi_part_gens(r=3)[0] == MF.E2()^3 * MF.Delta()
True

sage: MF = QuasiCuspForms(n=infinity, k=18, ep=-1)
sage: MF.quasi_part_gens(r=1, min_exp=-2) == MF.quasi_part_gens(r=1, min_
↪exp=1)
True
sage: MF.quasi_part_gens(r=1)
[q - 8*q^2 - 8*q^3 + 5952*q^4 + O(q^5),
 q^2 - 8*q^3 + 208*q^4 + O(q^5),
 q^3 - 16*q^4 + O(q^5)]

sage: MF = QuasiWeakModularForms(n=infinity, k=4, ep=1)
sage: MF.quasi_part_gens(r=2, min_exp=2, order_1=-2)[0] == MF.E2()^2 * MF.
↪E4()^(-2) * MF.f_inf()^2
True
sage: [v.order_at(-1) for v in MF.quasi_part_gens(r=0, min_exp=2, order_1=-2)]
[-2, -2]
```

rank ( )

Return the rank of `self` .

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.space import ModularForms
sage: MF = ModularForms(n=4, k=24, ep=-1)
sage: MF.rank()
3
sage: MF.subspace([MF.gen(0), MF.gen(2)]).rank()
2
```

rationalize_series (*laurent_series*, *coeff_bound=1e-10*, *denom_factor=1*)

Try to return a Laurent series with coefficients in `self.coeff_ring()` that matches the given Laurent series.

We give our best but there is absolutely no guarantee that it will work!

INPUT:

- **`laurent_series` – A Laurent series. If the Laurent coefficients already** coerce into `self.coeff_ring()` with a formal parameter then the Laurent series is returned as is.

  Otherwise it is assumed that the series is normalized in the sense that the first non-trivial coefficient is a power of `d` (e.g. `1`).

- **`coeff_bound` – Either `None` resp. `0` or a positive real number** (default: `1e-10`). If specified `coeff_bound` gives a lower bound for the size of the initial Laurent coefficients. If a coefficient is smaller it is assumed to be zero.

  For calculations with very small coefficients (less than `1e-10`) `coeff_bound` should be set to something even smaller or just `0` .

  Non-exact calculations often produce non-zero coefficients which are supposed to be zero. In those cases this parameter helps a lot.

- **`denom_factor` – An integer (default: 1) whose factor might occur in** the denominator of the given Laurent coefficients (in addition to naturally occuring factors).

OUTPUT:

A Laurent series over `self.coeff_ring()` corresponding to the given Laurent series.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.space import WeakModularForms,␣
↪ModularForms, QuasiCuspForms
sage: WF = WeakModularForms(n=14)
sage: qexp = WF.J_inv().q_expansion_fixed_d(d_num_prec=1000)
sage: qexp.parent()
Laurent Series Ring in q over Real Field with 1000 bits of precision
sage: qexp_int = WF.rationalize_series(qexp)
sage: qexp_int.add_bigoh(3)
d*q^-1 + 37/98 + 2587/(38416*d)*q + 899/(117649*d^2)*q^2 + O(q^3)
sage: qexp_int == WF.J_inv().q_expansion()
True
sage: WF.rationalize_series(qexp_int) == qexp_int
True
sage: WF(qexp_int) == WF.J_inv()
True

sage: WF.rationalize_series(qexp.parent()(1))
1
sage: WF.rationalize_series(qexp_int.parent()(1)).parent()
Laurent Series Ring in q over Fraction Field of Univariate Polynomial Ring in␣
↪d over Integer Ring

sage: MF = ModularForms(n=infinity, k=4)
sage: qexp = MF.E4().q_expansion_fixed_d()
sage: qexp.parent()
Power Series Ring in q over Rational Field
sage: qexp_int = MF.rationalize_series(qexp)
sage: qexp_int.parent()
Power Series Ring in q over Fraction Field of Univariate Polynomial Ring in d␣
↪over Integer Ring
sage: qexp_int == MF.E4().q_expansion()
True
sage: MF.rationalize_series(qexp_int) == qexp_int
True
sage: MF(qexp_int) == MF.E4()
True

sage: QF = QuasiCuspForms(n=8, k=22/3, ep=-1)
sage: el = QF(QF.f_inf()*QF.E2())
sage: qexp = el.q_expansion_fixed_d(d_num_prec=1000)
sage: qexp.parent()
```

```
Power Series Ring in q over Real Field with 1000 bits of precision
sage: qexp_int = QF.rationalize_series(qexp)
sage: qexp_int.parent()
Power Series Ring in q over Fraction Field of Univariate Polynomial Ring in d␣
↪over Integer Ring
sage: qexp_int == el.q_expansion()
True
sage: QF.rationalize_series(qexp_int) == qexp_int
True
sage: QF(qexp_int) == el
True
```

required_laurent_prec (*min_exp=0*, *order_1=0*)

Return an upper bound for the required precision for Laurent series to uniquely determine a corresponding (quasi) form in `self` with the given lower bound `min_exp` for the order at infinity (for each quasi part).

For `n=infinity` only the holomorphic case (`min_exp >= 0`) is supported (in particular a non-negative order at `-1` is assumed).

INPUT:

- **`min_exp` – An integer (default: 0), namely the lower bound for the** order at infinity resp. the exponent of the Laurent series.
- **`order_1` – A lower bound for the order at `-1` for all quasi parts** (default: 0). If `n!=infinity` this parameter is ignored.

OUTPUT:

An integer, namely an upper bound for the number of required Laurent coefficients. The bound should be precise or at least pretty sharp.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.space import␣
↪QuasiWeakModularForms, ModularForms, QuasiModularForms
sage: QF = QuasiWeakModularForms(n=8, k=10/3, ep=-1)
sage: QF.required_laurent_prec(min_exp=-1)
5

sage: MF = ModularForms(k=36)
sage: MF.required_laurent_prec(min_exp=2)
4

sage: QuasiModularForms(k=2).required_laurent_prec()
1

sage: QuasiWeakModularForms(n=infinity, k=2, ep=-1).required_laurent_
↪prec(order_1=-1)
6
```

subspace (*basis*)

Return the subspace of `self` generated by `basis` .

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.space import ModularForms
sage: MF = ModularForms(k=24)
sage: MF.dimension()
3
```

```
sage: subspace = MF.subspace([MF.gen(0), MF.gen(1)])
sage: subspace
Subspace of dimension 2 of ModularForms(n=3, k=24, ep=1) over Integer Ring
```

weight ( )

Return the weight of (elements of) `self` .

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.space import QuasiModularForms
sage: QuasiModularForms(n=16, k=16/7, ep=-1).weight()
16/7
```

weight_parameters ( )

Check whether `self` has a valid weight and multiplier.

If not then an exception is raised. Otherwise the two weight parameters corresponding to the weight and multiplier of `self` are returned.

The weight parameters are e.g. used to calculate dimensions or precisions of Fourier expansion.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.space import␣
↪MeromorphicModularForms
sage: MF = MeromorphicModularForms(n=18, k=-7, ep=-1)
sage: MF.weight_parameters()
(-3, 17)
sage: (MF._l1, MF._l2) == MF.weight_parameters()
True
sage: (k, ep) = (MF.weight(), MF.ep())
sage: n = MF.hecke_n()
sage: k == 4*(n*MF._l1 + MF._l2)/(n-2) + (1-ep)*n/(n-2)
True

sage: from sage.modular.modform_hecketriangle.space import ModularForms
sage: MF = ModularForms(n=5, k=12, ep=1)
sage: MF.weight_parameters()
(1, 4)
sage: (MF._l1, MF._l2) == MF.weight_parameters()
True
sage: (k, ep) = (MF.weight(), MF.ep())
sage: n = MF.hecke_n()
sage: k == 4*(n*MF._l1 + MF._l2)/(n-2) + (1-ep)*n/(n-2)
True
sage: MF.dimension() == MF._l1 + 1
True

sage: MF = ModularForms(n=infinity, k=8, ep=1)
sage: MF.weight_parameters()
(2, 0)
sage: MF.dimension() == MF._l1 + 1
True
```

CHAPTER FOUR

ELEMENTS OF HECKE MODULAR FORMS SPACES

AUTHORS:

- Jonas Jermann (2013): initial version

class `sage.modular.modform_hecketriangle.element.`**`FormsElement`**`(`*parent*, *rat*`)`

Bases: *`sage.modular.modform_hecketriangle.graded_ring_element.FormsRingElement`*

(Hecke) modular forms.

`ambient_coordinate_vector``()`

Return the coordinate vector of `self` with respect to `self.parent().ambient_space().gens()` .

The returned coordinate vector is an element of `self.parent().module()` .

Mote: This uses the corresponding function of the parent. If the parent has not defined a coordinate vector function or an ambient module for coordinate vectors then an exception is raised by the parent (default implementation).

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.space import ModularForms
sage: MF = ModularForms(n=4, k=24, ep=-1)
sage: MF.gen(0).ambient_coordinate_vector().parent()
Vector space of dimension 3 over Fraction Field of Univariate Polynomial Ring␣
↪in d over Integer Ring
sage: MF.gen(0).ambient_coordinate_vector()
(1, 0, 0)
sage: subspace = MF.subspace([MF.gen(0), MF.gen(2)])
sage: subspace.gen(0).ambient_coordinate_vector().parent()
Vector space of degree 3 and dimension 2 over Fraction Field of Univariate␣
↪Polynomial Ring in d over Integer Ring
Basis matrix:
[1 0 0]
[0 0 1]
sage: subspace.gen(0).ambient_coordinate_vector()
(1, 0, 0)
sage: subspace.gen(0).ambient_coordinate_vector() == subspace.ambient_
↪coordinate_vector(subspace.gen(0))
True
```

`coordinate_vector``()`

Return the coordinate vector of `self` with respect to `self.parent().gens()` .

This uses the corresponding function of the parent. If the parent has not defined a coordinate vector function or a module for coordinate vectors then an exception is raised by the parent (default implementation).

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.space import ModularForms
sage: MF = ModularForms(n=4, k=24, ep=-1)
sage: MF.gen(0).coordinate_vector().parent()
Vector space of dimension 3 over Fraction Field of Univariate Polynomial Ring␣
↪in d over Integer Ring
sage: MF.gen(0).coordinate_vector()
(1, 0, 0)
sage: subspace = MF.subspace([MF.gen(0), MF.gen(2)])
sage: subspace.gen(0).coordinate_vector().parent()
Vector space of dimension 2 over Fraction Field of Univariate Polynomial Ring␣
↪in d over Integer Ring
sage: subspace.gen(0).coordinate_vector()
(1, 0)
sage: subspace.gen(0).coordinate_vector() == subspace.coordinate_
↪vector(subspace.gen(0))
True
```

lseries (*num_prec=None*, *max_imaginary_part=0*, *max_asymp_coeffs=40*)

Return the L-series of `self` if `self` is modular and holomorphic. Note: This relies on the (pari) based function `Dokchitser`.

INPUT:

- **num_prec – An integer denoting the to-be-used numerical precision.** If integer `num_prec=None` (default) the default numerical precision of the parent of `self` is used.
- **max_imaginary_part – A real number (default: 0), indicating up to which** imaginary part the L-series is going to be studied.
- `max_asymp_coeffs` – An integer (default: 40).

OUTPUT:

An interface to Tim Dokchitser’s program for computing L-series, namely the series given by the Fourier coefficients of `self`.

EXAMPLES:

```
sage: from sage.modular.modform.eis_series import eisenstein_series_lseries
sage: from sage.modular.modform_hecketriangle.space import ModularForms
sage: f = ModularForms(n=3, k=4).E4()/240
sage: L = f.lseries()
sage: L
L-series associated to the modular form 1/240 + q + 9*q^2 + 28*q^3 + 73*q^4 +␣
↪O(q^5)
sage: L.conductor
1
sage: L(1).prec()
53
sage: L.check_functional_equation() < 2^(-50)
True
sage: L(1)
-0.0304484570583...
sage: abs(L(1) - eisenstein_series_lseries(4)(1)) < 2^(-53)
True
sage: L.derivative(1, 1)
-0.0504570844798...
sage: L.derivative(1, 2)/2
```

```
-0.0350657360354...
sage: L.taylor_series(1, 3)
-0.0304484570583... - 0.0504570844798...*z - 0.0350657360354...*z^2 + O(z^3)
sage: coeffs = f.q_expansion_vector(min_exp=0, max_exp=20, fix_d=True)
sage: sum([coeffs[k]*k^(-10) for k in range(1,len(coeffs))]).n(53)
1.00935215408...
sage: L(10)
1.00935215649...

sage: f = ModularForms(n=6, k=4).E4()
sage: L = f.lseries(num_prec=200)
sage: L.conductor
3
sage: L.check_functional_equation() < 2^(-180)
True
sage: L(1)
-2.92305187760575399490414692523085855811204642031749788...
sage: L(1).prec()
200
sage: coeffs = f.q_expansion_vector(min_exp=0, max_exp=20, fix_d=True)
sage: sum([coeffs[k]*k^(-10) for k in range(1,len(coeffs))]).n(53)
24.2281438789...
sage: L(10).n(53)
24.2281439447...

sage: f = ModularForms(n=8, k=6, ep=-1).E6()
sage: L = f.lseries()
sage: L.check_functional_equation() < 2^(-45)
True
sage: L.taylor_series(3, 3)
0.000000000000... + 0.867197036668...*z + 0.261129628199...*z^2 + O(z^3)
sage: coeffs = f.q_expansion_vector(min_exp=0, max_exp=20, fix_d=True)
sage: sum([coeffs[k]*k^(-10) for k in range(1,len(coeffs))]).n(53)
-13.0290002560...
sage: L(10).n(53)
-13.0290184579...

sage: f = (ModularForms(n=17, k=24).Delta()^2)     # long time
sage: L = f.lseries()     # long time
sage: L.check_functional_equation() < 2^(-50)     # long time
True
sage: L.taylor_series(12, 3)     # long time
0.000683924755280... - 0.000875942285963...*z + 0.000647618966023...*z^2 +␣
↪O(z^3)
sage: coeffs = f.q_expansion_vector(min_exp=0, max_exp=20, fix_d=True)     #␣
↪long time
sage: sum([coeffs[k]*k^(-30) for k in range(1,len(coeffs))]).n(53)     # long␣
↪time
9.31562890589...e-10
sage: L(30).n(53)     # long time
9.31562890589...e-10

sage: f = ModularForms(n=infinity, k=2, ep=-1).f_i()
sage: L = f.lseries()
sage: L.check_functional_equation() < 2^(-50)
True
sage: L.taylor_series(1, 3)
0.000000000000... + 5.76543616701...*z + 9.92776715593...*z^2 + O(z^3)
```

```
sage: coeffs = f.q_expansion_vector(min_exp=0, max_exp=20, fix_d=True)
sage: sum([coeffs[k]*k^(-10) for k in range(1,len(coeffs))]).n(53)
-23.9781792831...
sage: L(10).n(53)
-23.9781792831...
```

CHAPTER

FIVE

ELEMENTS OF GRADED RINGS OF MODULAR FORMS FOR HECKE TRIANGLE GROUPS

AUTHORS:

- Jonas Jermann (2013): initial version

class sage.modular.modform_hecketriangle.graded_ring_element.**FormsRingElement** (*parent*, *rat*)

Bases: sage.structure.element.CommutativeAlgebraElement, sage.structure.unique_representation.UniqueRepresentation

Element of a FormsRing.

AnalyticType

alias of *AnalyticType*

analytic_type ()

Return the analytic type of self.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.graded_ring import␣
↪QuasiMeromorphicModularFormsRing
sage: from sage.modular.modform_hecketriangle.space import␣
↪QuasiMeromorphicModularForms
sage: x,y,z,d = var("x,y,z,d")
sage: QuasiMeromorphicModularFormsRing(n=5)(x/z+d).analytic_type()
quasi meromorphic modular
sage: QuasiMeromorphicModularFormsRing(n=5)((y^3-z^5)/(x^5-y^2)+y-d).analytic_
↪type()
quasi weakly holomorphic modular
sage: QuasiMeromorphicModularFormsRing(n=5)(x^2+y-d).analytic_type()
modular
sage: QuasiMeromorphicModularForms(n=18).J_inv().analytic_type()
weakly holomorphic modular
sage: QuasiMeromorphicModularForms(n=18).f_inf().analytic_type()
cuspidal
sage: QuasiMeromorphicModularForms(n=infinity).f_inf().analytic_type()
modular
```

as_ring_element ()

Coerce self into the graded ring of its parent.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.space import CuspForms
sage: Delta = CuspForms(k=12).Delta()
```

```
sage: Delta.parent()
CuspForms(n=3, k=12, ep=1) over Integer Ring
sage: Delta.as_ring_element()
f_rho^3*d - f_i^2*d
sage: Delta.as_ring_element().parent()
CuspFormsRing(n=3) over Integer Ring

sage: CuspForms(n=infinity, k=12).Delta().as_ring_element()
-E4^2*f_i^2*d + E4^3*d
```

base_ring ( )

Return base ring of `self.parent()` .

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.space import ModularForms
sage: ModularForms(n=12, k=4, base_ring=CC).E4().base_ring()
Complex Field with 53 bits of precision
```

coeff_ring ( )

Return coefficient ring of `self` .

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.graded_ring import␣
↪ModularFormsRing
sage: ModularFormsRing().E6().coeff_ring()
Fraction Field of Univariate Polynomial Ring in d over Integer Ring
```

degree ( )

Return the degree of `self` in the graded ring. If `self` is not homogeneous, then `(None,None)` is returned.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.graded_ring import␣
↪QuasiModularFormsRing
sage: from sage.modular.modform_hecketriangle.space import ModularForms
sage: x,y,z,d = var("x,y,z,d")
sage: QuasiModularFormsRing()(x+y).degree() == (None, None)
True
sage: ModularForms(n=18).f_i().degree()
(9/4, -1)
sage: ModularForms(n=infinity).f_rho().degree()
(0, 1)
```

denominator ( )

Return the denominator of `self` . I.e. the (properly reduced) new form corresponding to the numerator of `self.rat()` .

Note that the parent of `self` might (probably will) change.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.graded_ring import␣
↪QuasiMeromorphicModularFormsRing
sage: from sage.modular.modform_hecketriangle.space import␣
↪QuasiMeromorphicModularForms
sage: x,y,z,d = var("x,y,z,d")
```

```
sage: QuasiMeromorphicModularFormsRing(n=5).Delta().full_reduce().
↪denominator()
1 + O(q^5)
sage: QuasiMeromorphicModularFormsRing(n=5)((y^3-z^5)/(x^5-y^2)+y-d).
↪denominator()
f_rho^5 - f_i^2
sage: QuasiMeromorphicModularFormsRing(n=5)((y^3-z^5)/(x^5-y^2)+y-d).
↪denominator().parent()
QuasiModularFormsRing(n=5) over Integer Ring
sage: QuasiMeromorphicModularForms(n=5, k=-2, ep=-1)(x/y).denominator()
1 - 13/(40*d)*q - 351/(64000*d^2)*q^2 - 13819/(76800000*d^3)*q^3 - 1163669/
↪(491520000000*d^4)*q^4 + O(q^5)
sage: QuasiMeromorphicModularForms(n=5, k=-2, ep=-1)(x/y).denominator().
↪parent()
QuasiModularForms(n=5, k=10/3, ep=-1) over Integer Ring
sage: (QuasiMeromorphicModularForms(n=infinity, k=-6, ep=-1)(y/(x*(x-y^2)))).
↪denominator()
-64*q - 512*q^2 - 768*q^3 + 4096*q^4 + O(q^5)
sage: (QuasiMeromorphicModularForms(n=infinity, k=-6, ep=-1)(y/(x*(x-y^2)))).
↪denominator().parent()
QuasiModularForms(n=+Infinity, k=8, ep=1) over Integer Ring
```

derivative ()

Return the derivative `d/dq = lambda/(2*pi*i) d/dtau` of `self`.

Note that the parent might (probably will) change. In particular its analytic type will be extended to contain "quasi".

If `parent.has_reduce_hom() == True` then the result is reduced to be an element of the corresponding forms space if possible.

In particular this is the case if `self` is a (homogeneous) element of a forms space.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.graded_ring import␣
↪QuasiMeromorphicModularFormsRing
sage: MR = QuasiMeromorphicModularFormsRing(n=7, red_hom=True)
sage: n = MR.hecke_n()
sage: E2 = MR.E2().full_reduce()
sage: E6 = MR.E6().full_reduce()
sage: f_rho = MR.f_rho().full_reduce()
sage: f_i = MR.f_i().full_reduce()
sage: f_inf = MR.f_inf().full_reduce()

sage: derivative(f_rho) == 1/n * (f_rho*E2 - f_i)
True
sage: derivative(f_i)   == 1/2 * (f_i*E2 - f_rho**(n-1))
True
sage: derivative(f_inf) == f_inf * E2
True
sage: derivative(f_inf).parent()
QuasiCuspForms(n=7, k=38/5, ep=-1) over Integer Ring
sage: derivative(E2)    == (n-2)/(4*n) * (E2**2 - f_rho**(n-2))
True
sage: derivative(E2).parent()
QuasiModularForms(n=7, k=4, ep=1) over Integer Ring

sage: MR = QuasiMeromorphicModularFormsRing(n=infinity, red_hom=True)
```

```
sage: E2 = MR.E2().full_reduce()
sage: E4 = MR.E4().full_reduce()
sage: E6 = MR.E6().full_reduce()
sage: f_i = MR.f_i().full_reduce()
sage: f_inf = MR.f_inf().full_reduce()

sage: derivative(E4)    == E4 * (E2 - f_i)
True
sage: derivative(f_i)   == 1/2 * (f_i*E2 - E4)
True
sage: derivative(f_inf) == f_inf * E2
True
sage: derivative(f_inf).parent()
QuasiModularForms(n=+Infinity, k=6, ep=-1) over Integer Ring
sage: derivative(E2)    == 1/4 * (E2**2 - E4)
True
sage: derivative(E2).parent()
QuasiModularForms(n=+Infinity, k=4, ep=1) over Integer Ring
```

diff_op (*op*, *new_parent=None*)

Return the differential operator `op` applied to `self`. If `parent.has_reduce_hom() == True` then the result is reduced to be an element of the corresponding forms space if possible.

INPUT:

- **op – An element of `self.parent().diff_alg()`.** I.e. an element of the algebra over `QQ` of differential operators generated by `X,Y,Z,dX,dY,DZ`, where e.g. `X` corresponds to the multiplication by `x` (resp. `f_rho`) and `dX` corresponds to `d/dx`.

  To expect a homogeneous result after applying the operator to a homogeneous element it should should be homogeneous operator (with respect to the usual, special grading).

- **new_parent – Try to convert the result to the specified** `new_parent`. If `new_parent == None` (default) then the parent is extended to a "quasi meromorphic" ring.

OUTPUT:

The new element.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.graded_ring import␣
↪QuasiMeromorphicModularFormsRing
sage: MR = QuasiMeromorphicModularFormsRing(n=8, red_hom=True)
sage: (X,Y,Z,dX,dY,dZ) = MR.diff_alg().gens()
sage: n=MR.hecke_n()
sage: mul_op = 4/(n-2)*X*dX + 2*n/(n-2)*Y*dY + 2*Z*dZ
sage: der_op = MR._derivative_op()
sage: ser_op = MR._serre_derivative_op()
sage: der_op == ser_op + (n-2)/(4*n)*Z*mul_op
True

sage: Delta = MR.Delta().full_reduce()
sage: E2 = MR.E2().full_reduce()
sage: Delta.diff_op(mul_op) == 12*Delta
True
sage: Delta.diff_op(mul_op).parent()
QuasiMeromorphicModularForms(n=8, k=12, ep=1) over Integer Ring
sage: Delta.diff_op(mul_op, Delta.parent()).parent()
CuspForms(n=8, k=12, ep=1) over Integer Ring
```

```
sage: E2.diff_op(mul_op, E2.parent()) == 2*E2
True
sage: Delta.diff_op(Z*mul_op, Delta.parent().extend_type("quasi", ring=True))␣
↪== 12*E2*Delta
True

sage: ran_op = X + Y*X*dY*dX + dZ + dX^2
sage: Delta.diff_op(ran_op)
f_rho^19*d + 306*f_rho^16*d - f_rho^11*f_i^2*d - 20*f_rho^10*f_i^2*d - 90*f_
↪rho^8*f_i^2*d
sage: E2.diff_op(ran_op)
f_rho*E2 + 1

sage: MR = QuasiMeromorphicModularFormsRing(n=infinity, red_hom=True)
sage: (X,Y,Z,dX,dY,dZ) = MR.diff_alg().gens()
sage: mul_op = 4*X*dX + 2*Y*dY + 2*Z*dZ
sage: der_op = MR._derivative_op()
sage: ser_op = MR._serre_derivative_op()
sage: der_op == ser_op + Z/4*mul_op
True

sage: Delta = MR.Delta().full_reduce()
sage: E2 = MR.E2().full_reduce()
sage: Delta.diff_op(mul_op) == 12*Delta
True
sage: Delta.diff_op(mul_op).parent()
QuasiMeromorphicModularForms(n=+Infinity, k=12, ep=1) over Integer Ring
sage: Delta.diff_op(mul_op, Delta.parent()).parent()
CuspForms(n=+Infinity, k=12, ep=1) over Integer Ring
sage: E2.diff_op(mul_op, E2.parent()) == 2*E2
True
sage: Delta.diff_op(Z*mul_op, Delta.parent().extend_type("quasi", ring=True))␣
↪== 12*E2*Delta
True

sage: ran_op = X + Y*X*dY*dX + dZ + dX^2
sage: Delta.diff_op(ran_op)
-E4^3*f_i^2*d + E4^4*d - 4*E4^2*f_i^2*d - 2*f_i^2*d + 6*E4*d
sage: E2.diff_op(ran_op)
E4*E2 + 1
```

ep ( )

Return the multiplier of `self`.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.graded_ring import␣
↪QuasiModularFormsRing
sage: from sage.modular.modform_hecketriangle.space import ModularForms
sage: x,y,z,d = var("x,y,z,d")
sage: QuasiModularFormsRing()(x+y).ep() is None
True
sage: ModularForms(n=18).f_i().ep()
-1
sage: ModularForms(n=infinity).E2().ep()
-1
```

evaluate (*tau*, *prec=None*, *num_prec=None*, *check=False*)

Try to return `self` evaluated at a point `tau` in the upper half plane, where `self` is interpreted as a function in `tau`, where `q=exp(2*pi*i*tau)`.

Note that this interpretation might not make sense (and fail) for certain (many) choices of (`base_ring`, `tau.parent()`).

It is possible to evalutate at points of `HyperbolicPlane()`. In this case the coordinates of the upper half plane model are used.

To obtain a precise and fast result the parameters `prec` and `num_prec` both have to be considered/balanced. A high `prec` value is usually quite costly.

INPUT:

- **`tau` – `infinity` or an element of the upper** half plane. E.g. with parent `AA` or `CC`.
- **`prec` – An integer, namely the precision used for the** Fourier expansion. If `prec == None` (default) then the default precision of `self.parent()` is used.
- **`num_prec` – An integer, namely the minimal numerical precision** used for `tau` and `d`. If `num_prec == None` (default) then the default numerical precision of `self.parent()` is used.
- **`check` – If `True` then the order of `tau` is checked.** Otherwise the order is only considered for `tau = infinity,i,rho,-1/rho`. Default: `False`.

OUTPUT:

The (numerical) evaluated function value.

ALGORITHM:

1. If the order of `self` at `tau` is known and nonzero: Return `0` resp. `infinity`.
2. Else if `tau==infinity` and the order is zero: Return the constant Fourier coefficient of `self`.
3. Else if `self` is homogeneous and modular:
   (a) Because of the (modular) transformation property of `self` the evaluation at `tau` is given by the evaluation at `w` multiplied by `aut_factor(A,w)`.
   (b) The evaluation at `w` is calculated by evaluating the truncated Fourier expansion of self at `q(w)`.

   Note that this is much faster and more precise than a direct evaluation at `tau`.
4. Else if `self` is exactly `E2`:
   (a) The same procedure as before is applied (with the aut_factor from the corresponding modular space).
   (b) Except that at the end a correction term for the quasimodular form `E2` of the form `4*lambda/(2*pi*i)*n/(n-2) * c*(c*w + d)` (resp. `4/(pi*i) * c*(c*w + d)` for `n=infinity`) has to be added, where `lambda = 2*cos(pi/n)` (resp `lambda = 2` for `n=infinity`) and `c,d` are the lower entries of the matrix `A`.
5. Else:
   (a) Evaluate `f_rho,f_i,E2` at `tau` using the above procedures. If `n=infinity` use `E4` instead of `f_rho`.
   (b) Substitute `x=f_rho(tau),y=f_i(tau),z=E2(tau)` and the numerical value of `d` for `d` in `self.rat()`. If `n=infinity` then subsitute `x=E4(tau)` instead.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.hecke_triangle_groups import␣
↪HeckeTriangleGroup
sage: from sage.modular.modform_hecketriangle.graded_ring import␣
↪QuasiMeromorphicModularFormsRing
sage: MR = QuasiMeromorphicModularFormsRing(n=5, red_hom=True)
sage: f_rho = MR.f_rho().full_reduce()
sage: f_i   = MR.f_i().full_reduce()
sage: f_inf = MR.f_inf().full_reduce()
sage: E2    = MR.E2().full_reduce()
sage: E4    = MR.E4().full_reduce()
sage: rho   = MR.group().rho()

sage: f_rho(rho)
0
sage: f_rho(rho + 1e-100)    # since rho == rho + 1e-100
0
sage: f_rho(rho + 1e-6)
2.525...e-10 - 3.884...e-6*I
sage: f_i(i)
0
sage: f_i(i + 1e-1000)
-6.084...e-14 - 4.101...e-1000*I
sage: f_inf(infinity)
0

sage: i = I = QuadraticField(-1, 'I').gen()
sage: z = -1/(-1/(2*i+30)-1)
sage: z
2/965*I + 934/965
sage: E4(z)
32288.05588811... - 118329.8566016...*I
sage: E4(z, prec=30, num_prec=100)    # long time
32288.0558872351130041311053... - 118329.856600349999751420381...*I
sage: E2(z)
409.3144737105... + 100.6926857489...*I
sage: E2(z, prec=30, num_prec=100)    # long time
409.314473710489761254584951... + 100.692685748952440684513866...*I
sage: (E2^2-E4)(z)
125111.2655383... + 200759.8039479...*I
sage: (E2^2-E4)(z, prec=30, num_prec=100)    # long time
125111.265538336196262200469... + 200759.803948009905410385699...*I

sage: (E2^2-E4)(infinity)
0
sage: (1/(E2^2-E4))(infinity)
+Infinity
sage: ((E2^2-E4)/f_inf)(infinity)
-3/(10*d)

sage: G = HeckeTriangleGroup(n=8)
sage: MR = QuasiMeromorphicModularFormsRing(group=G, red_hom=True)
sage: f_rho = MR.f_rho().full_reduce()
sage: f_i   = MR.f_i().full_reduce()
sage: E2    = MR.E2().full_reduce()

sage: z = AlgebraicField()(1/10+13/10*I)
sage: A = G.V(4)
sage: S = G.S()
```

```
sage: T = G.T()
sage: A == (T*S)**3*T
True
sage: az = A.acton(z)
sage: az == (A[0,0]*z + A[0,1]) / (A[1,0]*z + A[1,1])
True

sage: f_rho(z)
1.03740476727... + 0.0131941034523...*I
sage: f_rho(az)
-2.29216470688... - 1.46235057536...*I
sage: k = f_rho.weight()
sage: aut_fact = f_rho.ep()^3 * (((T*S)**2*T).acton(z)/
↪AlgebraicField()(i))**k * (((T*S)*T).acton(z)/AlgebraicField()(i))**k * (T.
↪acton(z)/AlgebraicField()(i))**k
sage: abs(aut_fact - f_rho.parent().aut_factor(A, z)) < 1e-12
True
sage: aut_fact * f_rho(z)
-2.29216470688... - 1.46235057536...*I

sage: f_rho.parent().default_num_prec(1000)
sage: f_rho.parent().default_prec(300)
sage: (f_rho.q_expansion_fixed_d().polynomial())(exp((2*pi*i).n(1000)*z/G.
↪lam()))    # long time
1.0374047672719462149821251... + 0.013194103452368974597290332...*I
sage: (f_rho.q_expansion_fixed_d().polynomial())(exp((2*pi*i).n(1000)*az/G.
↪lam()))    # long time
-2.2921647068881834598616367... - 1.4623505753697635207183406...*I

sage: f_i(z)
0.667489320423... - 0.118902824870...*I
sage: f_i(az)
14.5845388476... - 28.4604652892...*I
sage: k = f_i.weight()
sage: aut_fact = f_i.ep()^3 * (((T*S)**2*T).acton(z)/AlgebraicField()(i))**k␣
↪* (((T*S)*T).acton(z)/AlgebraicField()(i))**k * (T.acton(z)/
↪AlgebraicField()(i))**k
sage: abs(aut_fact - f_i.parent().aut_factor(A, z)) < 1e-12
True
sage: aut_fact * f_i(z)
14.5845388476... - 28.4604652892...*I

sage: f_i.parent().default_num_prec(1000)
sage: f_i.parent().default_prec(300)
sage: (f_i.q_expansion_fixed_d().polynomial())(exp((2*pi*i).n(1000)*z/G.
↪lam()))    # long time
0.66748932042300250077433252... - 0.11890282487028677063054267...*I
sage: (f_i.q_expansion_fixed_d().polynomial())(exp((2*pi*i).n(1000)*az/G.
↪lam()))    # long time
14.584538847698600875918891... - 28.460465289220303834894855...*I

sage: f = f_rho*E2
sage: f(z)
0.966024386418... - 0.0138894699429...*I
sage: f(az)
-15.9978074989... - 29.2775758341...*I
sage: k = f.weight()
sage: aut_fact = f.ep()^3 * (((T*S)**2*T).acton(z)/AlgebraicField()(i))**k *␣
↪(((T*S)*T).acton(z)/AlgebraicField()(i))**k * (T.acton(z)/
↪AlgebraicField()(i))**k
```

```
sage: abs(aut_fact - f.parent().aut_factor(A, z)) < 1e-12
True
sage: k2 = f_rho.weight()
sage: aut_fact2 = f_rho.ep() * (((T*S)**2*T).acton(z)/
↪AlgebraicField()(i))**k2 * (((T*S)*T).acton(z)/AlgebraicField()(i))**k2 *␣
↪(T.acton(z)/AlgebraicField()(i))**k2
sage: abs(aut_fact2 - f_rho.parent().aut_factor(A, z)) < 1e-12
True
sage: cor_term = (4 * G.n() / (G.n()-2) * A.c() * (A.c()*z+A.d())) / (2*pi*i).
↪n(1000) * G.lam()
sage: aut_fact*f(z) + cor_term*aut_fact2*f_rho(z)
-15.9978074989... - 29.2775758341...*I

sage: f.parent().default_num_prec(1000)
sage: f.parent().default_prec(300)
sage: (f.q_expansion_fixed_d().polynomial())(exp((2*pi*i).n(1000)*z/G.lam()))␣
↪   # long time
0.966024386418672967778094 36... - 0.0138894699429955308073115 03...*I
sage: (f.q_expansion_fixed_d().polynomial())(exp((2*pi*i).n(1000)*az/G.
↪lam()))    # long time
-15.9978074989588253528870 40... - 29.2775758341232460634322 06...*I

sage: MR = QuasiMeromorphicModularFormsRing(n=infinity, red_hom=True)
sage: f_i   = MR.f_i().full_reduce()
sage: f_inf = MR.f_inf().full_reduce()
sage: E2    = MR.E2().full_reduce()
sage: E4    = MR.E4().full_reduce()

sage: f_i(i)
0
sage: f_i(i + 1e-1000)
2.991...e-12 - 3.048...e-1000*I
sage: f_inf(infinity)
0

sage: z = -1/(-1/(2*i+30)-1)
sage: E4(z, prec=15)
804.0722034... + 211.9278206...*I
sage: E4(z, prec=30, num_prec=100)    # long time
803.928382417... + 211.889914044...*I
sage: E2(z)
2.438455612... - 39.48442265...*I
sage: E2(z, prec=30, num_prec=100)    # long time
2.43968197227756036957475... - 39.4842637577742677851431...*I
sage: (E2^2-E4)(z)
-2265.442515... - 380.3197877...*I
sage: (E2^2-E4)(z, prec=30, num_prec=100)    # long time
-2265.44251550679807447320... - 380.319787790548788238792...*I

sage: (E2^2-E4)(infinity)
0
sage: (1/(E2^2-E4))(infinity)
+Infinity
sage: ((E2^2-E4)/f_inf)(infinity)
-1/(2*d)

sage: G = HeckeTriangleGroup(n=Infinity)
sage: z = AlgebraicField()(1/10+13/10*I)
```

```
sage: A = G.V(4)
sage: S = G.S()
sage: T = G.T()
sage: A == (T*S)**3*T
True
sage: az = A.acton(z)
sage: az == (A[0,0]*z + A[0,1]) / (A[1,0]*z + A[1,1])
True

sage: f_i(z)
0.6208853409... - 0.1212525492...*I
sage: f_i(az)
6.103314419... + 20.42678597...*I
sage: k = f_i.weight()
sage: aut_fact = f_i.ep()^3 * (((T*S)**2*T).acton(z)/AlgebraicField()(i))**k␣
↪* (((T*S)*T).acton(z)/AlgebraicField()(i))**k * (T.acton(z)/
↪AlgebraicField()(i))**k
sage: abs(aut_fact - f_i.parent().aut_factor(A, z)) < 1e-12
True
sage: aut_fact * f_i(z)
6.103314419... + 20.42678597...*I

sage: f_i.parent().default_num_prec(1000)
sage: f_i.parent().default_prec(300)
sage: (f_i.q_expansion_fixed_d().polynomial())(exp((2*pi*i).n(1000)*z/G.
↪lam()))    # long time
0.620885340917559158572271... - 0.121252549240996430425967...*I
sage: (f_i.q_expansion_fixed_d().polynomial())(exp((2*pi*i).n(1000)*az/G.
↪lam()))    # long time
6.103314419751981867450l7... + 20.4267859728657976382684...*I

sage: f = f_i*E2
sage: f(z)
0.5349190275... - 0.1322370856...*I
sage: f(az)
-140.4711702... + 469.0793692...*I
sage: k = f.weight()
sage: aut_fact = f.ep()^3 * (((T*S)**2*T).acton(z)/AlgebraicField()(i))**k *␣
↪(((T*S)*T).acton(z)/AlgebraicField()(i))**k * (T.acton(z)/
↪AlgebraicField()(i))**k
sage: abs(aut_fact - f.parent().aut_factor(A, z)) < 1e-12
True
sage: k2 = f_i.weight()
sage: aut_fact2 = f_i.ep() * (((T*S)**2*T).acton(z)/AlgebraicField()(i))**k2␣
↪* (((T*S)*T).acton(z)/AlgebraicField()(i))**k2 * (T.acton(z)/
↪AlgebraicField()(i))**k2
sage: abs(aut_fact2 - f_i.parent().aut_factor(A, z)) < 1e-12
True
sage: cor_term = (4 * A.c() * (A.c()*z+A.d())) / (2*pi*i).n(1000) * G.lam()
sage: aut_fact*f(z) + cor_term*aut_fact2*f_i(z)
-140.4711702... + 469.0793692...*I

sage: f.parent().default_num_prec(1000)
sage: f.parent().default_prec(300)
sage: (f.q_expansion_fixed_d().polynomial())(exp((2*pi*i).n(1000)*z/G.lam()))␣
↪    # long time
0.534919027587592616802582... - 0.132237085641931661668338...*I
```

```
sage: (f.q_expansion_fixed_d().polynomial())(exp((2*pi*i).n(1000)*az/G.
↪lam()))     # long time
-140.471170232432551196978... + 469.079369280804086032719...*I
```

It is possible to evaluate at points of `HyperbolicPlane()` :

```
sage: p = HyperbolicPlane().PD().get_point(-I/2)
sage: bool(p.to_model('UHP').coordinates() == I/3)
True
sage: E4(p) == E4(I/3)
True
sage: p = HyperbolicPlane().PD().get_point(I)
sage: f_inf(p, check=True) == 0
True
sage: (1/(E2^2-E4))(p) == infinity
True
```

full_reduce ( )

Convert `self` into its reduced parent.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.graded_ring import␣
↪QuasiMeromorphicModularFormsRing
sage: Delta = QuasiMeromorphicModularFormsRing().Delta()
sage: Delta
f_rho^3*d - f_i^2*d
sage: Delta.full_reduce()
q - 24*q^2 + 252*q^3 - 1472*q^4 + O(q^5)
sage: Delta.full_reduce().parent() == Delta.reduced_parent()
True

sage: QuasiMeromorphicModularFormsRing().Delta().full_reduce().parent()
CuspForms(n=3, k=12, ep=1) over Integer Ring
```

group ( )

Return the (Hecke triangle) group of `self.parent()` .

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.space import ModularForms
sage: ModularForms(n=12, k=4).E4().group()
Hecke triangle group for n = 12
```

hecke_n ( )

Return the parameter `n` of the (Hecke triangle) group of `self.parent()` .

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.space import ModularForms
sage: ModularForms(n=12, k=6).E6().hecke_n()
12
```

is_cuspidal ( )

Return whether `self` is cuspidal in the sense that `self` is holomorphic and `f_inf` divides the numerator.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.graded_ring import␣
↪QuasiModularFormsRing
sage: from sage.modular.modform_hecketriangle.space import QuasiModularForms
sage: x,y,z,d = var("x,y,z,d")
sage: QuasiModularFormsRing(n=5)(y^3-z^5).is_cuspidal()
False
sage: QuasiModularFormsRing(n=5)(z*x^5-z*y^2).is_cuspidal()
True
sage: QuasiModularForms(n=18).Delta().is_cuspidal()
True
sage: QuasiModularForms(n=18).f_rho().is_cuspidal()
False
sage: QuasiModularForms(n=infinity).f_inf().is_cuspidal()
False
sage: QuasiModularForms(n=infinity).Delta().is_cuspidal()
True
```

is_holomorphic ( )

Return whether `self` is holomorphic in the sense that the denominator of `self` is constant.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.graded_ring import␣
↪QuasiMeromorphicModularFormsRing
sage: from sage.modular.modform_hecketriangle.space import␣
↪QuasiMeromorphicModularForms
sage: x,y,z,d = var("x,y,z,d")
sage: QuasiMeromorphicModularFormsRing(n=5)((y^3-z^5)/(x^5-y^2)+y-d).is_
↪holomorphic()
False
sage: QuasiMeromorphicModularFormsRing(n=5)(x^2+y-d+z).is_holomorphic()
True
sage: QuasiMeromorphicModularForms(n=18).J_inv().is_holomorphic()
False
sage: QuasiMeromorphicModularForms(n=18).f_i().is_holomorphic()
True
sage: QuasiMeromorphicModularForms(n=infinity).f_inf().is_holomorphic()
True
```

is_homogeneous ( )

Return whether `self` is homogeneous.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.graded_ring import␣
↪QuasiModularFormsRing
sage: QuasiModularFormsRing(n=12).Delta().is_homogeneous()
True
sage: QuasiModularFormsRing(n=12).Delta().parent().is_homogeneous()
False
sage: x,y,z,d=var("x,y,z,d")
sage: QuasiModularFormsRing(n=12)(x^3+y^2+z+d).is_homogeneous()
False

sage: QuasiModularFormsRing(n=infinity)(x*(x-y^2)+y^4).is_homogeneous()
True
```

is_modular ( )

Return whether `self` (resp. its homogeneous components) transform like modular forms.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.graded_ring import␣
↪QuasiModularFormsRing
sage: from sage.modular.modform_hecketriangle.space import QuasiModularForms
sage: x,y,z,d = var("x,y,z,d")
sage: QuasiModularFormsRing(n=5)(x^2+y-d).is_modular()
True
sage: QuasiModularFormsRing(n=5)(x^2+y-d+z).is_modular()
False
sage: QuasiModularForms(n=18).f_i().is_modular()
True
sage: QuasiModularForms(n=18).E2().is_modular()
False
sage: QuasiModularForms(n=infinity).f_inf().is_modular()
True
```

is_weakly_holomorphic ()

Return whether `self` is weakly holomorphic in the sense that: `self` has at most a power of `f_inf` in its denominator.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.graded_ring import␣
↪QuasiMeromorphicModularFormsRing
sage: from sage.modular.modform_hecketriangle.space import␣
↪QuasiMeromorphicModularForms
sage: x,y,z,d = var("x,y,z,d")
sage: QuasiMeromorphicModularFormsRing(n=5)(x/(x^5-y^2)+z).is_weakly_
↪holomorphic()
True
sage: QuasiMeromorphicModularFormsRing(n=5)(x^2+y/x-d).is_weakly_holomorphic()
False
sage: QuasiMeromorphicModularForms(n=18).J_inv().is_weakly_holomorphic()
True
sage: QuasiMeromorphicModularForms(n=infinity, k=-4)(1/x).is_weakly_
↪holomorphic()
True
sage: QuasiMeromorphicModularForms(n=infinity, k=-2)(1/y).is_weakly_
↪holomorphic()
False
```

is_zero ()

Return whether `self` is the zero function.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.graded_ring import␣
↪QuasiModularFormsRing
sage: from sage.modular.modform_hecketriangle.space import QuasiModularForms
sage: x,y,z,d = var("x,y,z,d")
sage: QuasiModularFormsRing(n=5)(1).is_zero()
False
sage: QuasiModularFormsRing(n=5)(0).is_zero()
True
sage: QuasiModularForms(n=18).zero().is_zero()
True
sage: QuasiModularForms(n=18).Delta().is_zero()
False
```

```
sage: QuasiModularForms(n=infinity).f_rho().is_zero()
False
```

numerator ( )

Return the numerator of `self`. I.e. the (properly reduced) new form corresponding to the numerator of `self.rat()`.

Note that the parent of `self` might (probably will) change.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.graded_ring import␣
↪QuasiMeromorphicModularFormsRing
sage: from sage.modular.modform_hecketriangle.space import␣
↪QuasiMeromorphicModularForms
sage: x,y,z,d = var("x,y,z,d")
sage: QuasiMeromorphicModularFormsRing(n=5)((y^3-z^5)/(x^5-y^2)+y-d).
↪numerator()
f_rho^5*f_i - f_rho^5*d - E2^5 + f_i^2*d
sage: QuasiMeromorphicModularFormsRing(n=5)((y^3-z^5)/(x^5-y^2)+y-d).
↪numerator().parent()
QuasiModularFormsRing(n=5) over Integer Ring
sage: QuasiMeromorphicModularForms(n=5, k=-2, ep=-1)(x/y).numerator()
1 + 7/(100*d)*q + 21/(160000*d^2)*q^2 + 1043/(192000000*d^3)*q^3 + 45479/
↪(1228800000000*d^4)*q^4 + O(q^5)
sage: QuasiMeromorphicModularForms(n=5, k=-2, ep=-1)(x/y).numerator().parent()
QuasiModularForms(n=5, k=4/3, ep=1) over Integer Ring
sage: (QuasiMeromorphicModularForms(n=infinity, k=-2, ep=-1)(y/x)).numerator()
1 - 24*q + 24*q^2 - 96*q^3 + 24*q^4 + O(q^5)
sage: (QuasiMeromorphicModularForms(n=infinity, k=-2, ep=-1)(y/x)).
↪numerator().parent()
QuasiModularForms(n=+Infinity, k=2, ep=-1) over Integer Ring
```

order_at (*tau=+Infinity*)

Return the (overall) order of `self` at `tau` if easily possible: Namely if `tau` is `infinity` or congruent to `i` resp. `rho`.

It is possible to determine the order of points from `HyperbolicPlane()`. In this case the coordinates of the upper half plane model are used.

If `self` is homogeneous and modular then the rational function `self.rat()` is used. Otherwise only `tau=infinity` is supported by using the Fourier expansion with increasing precision (until the order can be determined).

The function is mainly used to be able to work with the correct precision for Laurent series.

For quasi forms one cannot deduce the analytic type from this order at `infinity` since the analytic order is defined by the behavior on each quasi part and not by their linear combination.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.graded_ring import␣
↪QuasiMeromorphicModularFormsRing
sage: MR = QuasiMeromorphicModularFormsRing(red_hom=True)
sage: (MR.Delta()^3).order_at(infinity)
3
sage: MR.E2().order_at(infinity)
0
sage: (MR.J_inv()^2).order_at(infinity)
-2
```

```
sage: x,y,z,d = MR.pol_ring().gens()
sage: el = MR((z^3-y)^2/(x^3-y^2)).full_reduce()
sage: el
108*q + 11664*q^2 + 502848*q^3 + 12010464*q^4 + O(q^5)
sage: el.order_at(infinity)
1
sage: el.parent()
QuasiWeakModularForms(n=3, k=0, ep=1) over Integer Ring
sage: el.is_holomorphic()
False
sage: MR((z-y)^2+(x-y)^3).order_at(infinity)
2
sage: MR((x-y)^10).order_at(infinity)
10
sage: MR.zero().order_at(infinity)
+Infinity
sage: (MR(x*y^2)/MR.J_inv()).order_at(i)
2
sage: (MR(x*y^2)/MR.J_inv()).order_at(MR.group().rho())
-2

sage: MR = QuasiMeromorphicModularFormsRing(n=infinity, red_hom=True)
sage: (MR.Delta()^3*MR.E4()).order_at(infinity)
3
sage: MR.E2().order_at(infinity)
0
sage: (MR.J_inv()^2/MR.E4()).order_at(infinity)
-2
sage: el = MR((z^3-x*y)^2/(x^2*(x-y^2))).full_reduce()
sage: el
4*q - 304*q^2 + 8128*q^3 - 106144*q^4 + O(q^5)
sage: el.order_at(infinity)
1
sage: el.parent()
QuasiWeakModularForms(n=+Infinity, k=0, ep=1) over Integer Ring
sage: el.is_holomorphic()
False
sage: MR((z-x)^2+(x-y)^3).order_at(infinity)
2
sage: MR((x-y)^10).order_at(infinity)
10
sage: MR.zero().order_at(infinity)
+Infinity

sage: (MR.j_inv()*MR.f_i()^3).order_at(-1)
1
sage: (MR.j_inv()*MR.f_i()^3).order_at(i)
3
sage: (1/MR.f_inf()^2).order_at(-1)
0

sage: p = HyperbolicPlane().PD().get_point(I)
sage: MR((x-y)^10).order_at(p)
10
sage: MR.zero().order_at(p)
+Infinity
```

q_expansion (*prec=None*, *fix_d=False*, *d_num_prec=None*, *fix_prec=False*)

Returns the Fourier expansion of self.

INPUT:

- **prec – An integer, the desired output precision O(q^prec).** Default: `None` in which case the default precision of `self.parent()` is used.
- **fix_d – If False (default) a formal parameter is used for d.** If `True` then the numerical value of `d` is used (resp. an exact value if the group is arithmetic). Otherwise the given value is used for `d`.
- **d_num_prec – The precision to be used if a numerical value for d is substituted.** Default: `None` in which case the default numerical precision of `self.parent()` is used.
- **fix_prec – If fix_prec is not False (default)** then the precision of the `MFSeriesConstructor` is increased such that the output has exactly the specified precision O(q^prec).

OUTPUT:

The Fourier expansion of self as a `FormalPowerSeries` or `FormalLaurentSeries`.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.graded_ring import␣
↪WeakModularFormsRing, QuasiModularFormsRing
sage: j_inv = WeakModularFormsRing(red_hom=True).j_inv()
sage: j_inv.q_expansion(prec=3)
q^-1 + 31/(72*d) + 1823/(27648*d^2)*q + 10495/(2519424*d^3)*q^2 + O(q^3)

sage: E2 = QuasiModularFormsRing(n=5, red_hom=True).E2()
sage: E2.q_expansion(prec=3)
1 - 9/(200*d)*q - 369/(320000*d^2)*q^2 + O(q^3)
sage: E2.q_expansion(prec=3, fix_d=1)
1 - 9/200*q - 369/320000*q^2 + O(q^3)

sage: E6 = WeakModularFormsRing(n=5, red_hom=True).E6().full_reduce()
sage: Delta = WeakModularFormsRing(n=5, red_hom=True).Delta().full_reduce()
sage: E6.q_expansion(prec=3).prec() == 3
True
sage: (Delta/(E2^3-E6)).q_expansion(prec=3).prec() == 3
True
sage: (Delta/(E2^3-E6)^3).q_expansion(prec=3).prec() == 3
True
sage: ((E2^3-E6)/Delta^2).q_expansion(prec=3).prec() == 3
True
sage: ((E2^3-E6)^3/Delta).q_expansion(prec=3).prec() == 3
True

sage: x,y = var("x,y")
sage: el = WeakModularFormsRing()((x+1)/(x^3-y^2))
sage: el.q_expansion(prec=2, fix_prec = True)
2*d*q^-1 + O(1)
sage: el.q_expansion(prec=2)
2*d*q^-1 + 1/6 + 119/(41472*d)*q + O(q^2)

sage: j_inv = WeakModularFormsRing(n=infinity, red_hom=True).j_inv()
sage: j_inv.q_expansion(prec=3)
q^-1 + 3/(8*d) + 69/(1024*d^2)*q + 1/(128*d^3)*q^2 + O(q^3)

sage: E2 = QuasiModularFormsRing(n=infinity, red_hom=True).E2()
```

```
sage: E2.q_expansion(prec=3)
1 - 1/(8*d)*q - 1/(512*d^2)*q^2 + O(q^3)
sage: E2.q_expansion(prec=3, fix_d=1)
1 - 1/8*q - 1/512*q^2 + O(q^3)

sage: E4 = WeakModularFormsRing(n=infinity, red_hom=True).E4().full_reduce()
sage: Delta = WeakModularFormsRing(n=infinity, red_hom=True).Delta().full_
↪reduce()
sage: E4.q_expansion(prec=3).prec() == 3
True
sage: (Delta/(E2^2-E4)).q_expansion(prec=3).prec() == 3
True
sage: (Delta/(E2^2-E4)^3).q_expansion(prec=3).prec() == 3
True
sage: ((E2^2-E4)/Delta^2).q_expansion(prec=3).prec() == 3
True
sage: ((E2^2-E4)^3/Delta).q_expansion(prec=3).prec() == 3
True

sage: x,y = var("x,y")
sage: el = WeakModularFormsRing(n=infinity)((x+1)/(x-y^2))
sage: el.q_expansion(prec=2, fix_prec = True)
2*d*q^-1 + O(1)
sage: el.q_expansion(prec=2)
2*d*q^-1 + 1/2 + 39/(512*d)*q + O(q^2)
```

q_expansion_fixed_d (*prec=None*, *d_num_prec=None*, *fix_prec=False*)

Returns the Fourier expansion of self. The numerical (or exact) value for `d` is substituted.

INPUT:

- **prec – An integer, the desired output precision O(q^prec).** Default: `None` in which case the default precision of `self.parent()` is used.
- **d_num_prec – The precision to be used if a numerical value for d is substituted.** Default: `None` in which case the default numerical precision of `self.parent()` is used.
- **fix_prec – If fix_prec is not False (default)** then the precision of the `MFSeriesConstructor` is increased such that the output has exactly the specified precision O(q^prec).

OUTPUT:

The Fourier expansion of self as a `FormalPowerSeries` or `FormalLaurentSeries`.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.graded_ring import␣
↪WeakModularFormsRing, QuasiModularFormsRing
sage: j_inv = WeakModularFormsRing(red_hom=True).j_inv()
sage: j_inv.q_expansion_fixed_d(prec=3)
q^-1 + 744 + 196884*q + 21493760*q^2 + O(q^3)

sage: E2 = QuasiModularFormsRing(n=5, red_hom=True).E2()
sage: E2.q_expansion_fixed_d(prec=3)
1.000000000000... - 6.380956565426...*q - 23.18584547617...*q^2 + O(q^3)

sage: x,y = var("x,y")
sage: WeakModularFormsRing()((x+1)/(x^3-y^2)).q_expansion_fixed_d(prec=2, fix_
↪prec = True)
```

```
1/864*q^-1 + O(1)
sage: WeakModularFormsRing()((x+1)/(x^3-y^2)).q_expansion_fixed_d(prec=2)
1/864*q^-1 + 1/6 + 119/24*q + O(q^2)

sage: j_inv = WeakModularFormsRing(n=infinity, red_hom=True).j_inv()
sage: j_inv.q_expansion_fixed_d(prec=3)
q^-1 + 24 + 276*q + 2048*q^2 + O(q^3)

sage: E2 = QuasiModularFormsRing(n=infinity, red_hom=True).E2()
sage: E2.q_expansion_fixed_d(prec=3)
1 - 8*q - 8*q^2 + O(q^3)

sage: x,y = var("x,y")
sage: WeakModularFormsRing(n=infinity)((x+1)/(x-y^2)).q_expansion_fixed_
↪d(prec=2, fix_prec = True)
1/32*q^-1 + O(1)
sage: WeakModularFormsRing(n=infinity)((x+1)/(x-y^2)).q_expansion_fixed_
↪d(prec=2)
1/32*q^-1 + 1/2 + 39/8*q + O(q^2)

sage: (WeakModularFormsRing(n=14).J_inv()^3).q_expansion_fixed_d(prec=2)
2.933373093...e-6*q^-3 + 0.0002320999814...*q^-2 + 0.009013529265...*q^-1 + 0.
↪2292916854... + 4.303583833...*q + O(q^2)
```

q_expansion_vector (*min_exp=None*, *max_exp=None*, *prec=None*, ***kwargs*)

Return (part of) the Laurent series expansion of `self` as a vector.

INPUT:

- **min_exp – An integer, specifying the first coefficient to be** used for the vector. Default: `None`, meaning that the first non-trivial coefficient is used.
- **max_exp – An integer, specifying the last coefficient to be** used for the vector. Default: `None`, meaning that the default precision + 1 is used.
- **prec – An integer, specifying the precision of the underlying** Laurent series. Default: `None`, meaning that `max_exp + 1` is used.

OUTPUT:

A vector of size `max_exp -min_exp` over the coefficient ring of self, determined by the corresponding Laurent series coefficients.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.graded_ring import␣
↪WeakModularFormsRing
sage: f = WeakModularFormsRing(red_hom=True).j_inv()^3
sage: f.q_expansion(prec=3)
q^-3 + 31/(24*d)*q^-2 + 20845/(27648*d^2)*q^-1 + 7058345/(26873856*d^3) +␣
↪30098784355/(495338913792*d^4)*q + 175372747465/(17832200896512*d^5)*q^2 +␣
↪O(q^3)
sage: v = f.q_expansion_vector(max_exp=1, prec=3)
sage: v
(1, 31/(24*d), 20845/(27648*d^2), 7058345/(26873856*d^3), 30098784355/
↪(495338913792*d^4))
sage: v.parent()
Vector space of dimension 5 over Fraction Field of Univariate Polynomial Ring␣
↪in d over Integer Ring
sage: f.q_expansion_vector(min_exp=1, max_exp=2)
```

```
(30098784355/(495338913792*d^4), 175372747465/(17832200896512*d^5))
sage: f.q_expansion_vector(min_exp=1, max_exp=2, fix_d=True)
(541778118390, 151522053809760)

sage: f = WeakModularFormsRing(n=infinity, red_hom=True).j_inv()^3
sage: f.q_expansion_fixed_d(prec=3)
q^-3 + 72*q^-2 + 2556*q^-1 + 59712 + 1033974*q + 14175648*q^2 + O(q^3)
sage: v = f.q_expansion_vector(max_exp=1, prec=3, fix_d=True)
sage: v
(1, 72, 2556, 59712, 1033974)
sage: v.parent()
Vector space of dimension 5 over Rational Field
sage: f.q_expansion_vector(min_exp=1, max_exp=2)
(516987/(8388608*d^4), 442989/(33554432*d^5))
```

rat ( )

Return the rational function representing `self`.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.graded_ring import␣
↪ModularFormsRing
sage: ModularFormsRing(n=12).Delta().rat()
x^30*d - x^18*y^2*d
```

reduce (*force=False*)

In case `self.parent().has_reduce_hom() == True` (or `force==True`) and `self` is homogeneous the converted element lying in the corresponding homogeneous_part is returned.

Otherwise `self` is returned.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.graded_ring import␣
↪ModularFormsRing
sage: E2 = ModularFormsRing(n=7).E2().reduce()
sage: E2.parent()
QuasiModularFormsRing(n=7) over Integer Ring
sage: E2 = ModularFormsRing(n=7, red_hom=True).E2().reduce()
sage: E2.parent()
QuasiModularForms(n=7, k=2, ep=-1) over Integer Ring
sage: ModularFormsRing(n=7)(x+1).reduce().parent()
ModularFormsRing(n=7) over Integer Ring
sage: E2 = ModularFormsRing(n=7).E2().reduce(force=True)
sage: E2.parent()
QuasiModularForms(n=7, k=2, ep=-1) over Integer Ring
sage: ModularFormsRing(n=7)(x+1).reduce(force=True).parent()
ModularFormsRing(n=7) over Integer Ring

sage: y=var("y")
sage: ModularFormsRing(n=infinity)(x-y^2).reduce(force=True)
64*q - 512*q^2 + 1792*q^3 - 4096*q^4 + O(q^5)
```

reduced_parent ( )

Return the space with the analytic type of `self`. If `self` is homogeneous the corresponding `FormsSpace` is returned.

I.e. return the smallest known ambient space of `self`.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.graded_ring import␣
↪QuasiMeromorphicModularFormsRing
sage: Delta = QuasiMeromorphicModularFormsRing(n=7).Delta()
sage: Delta.parent()
QuasiMeromorphicModularFormsRing(n=7) over Integer Ring
sage: Delta.reduced_parent()
CuspForms(n=7, k=12, ep=1) over Integer Ring
sage: el = QuasiMeromorphicModularFormsRing()(x+1)
sage: el.parent()
QuasiMeromorphicModularFormsRing(n=3) over Integer Ring
sage: el.reduced_parent()
ModularFormsRing(n=3) over Integer Ring

sage: y=var("y")
sage: QuasiMeromorphicModularFormsRing(n=infinity)(x-y^2).reduced_parent()
ModularForms(n=+Infinity, k=4, ep=1) over Integer Ring
sage: QuasiMeromorphicModularFormsRing(n=infinity)(x*(x-y^2)).reduced_parent()
CuspForms(n=+Infinity, k=8, ep=1) over Integer Ring
```

serre_derivative ( )

Return the Serre derivative of `self`.

Note that the parent might (probably will) change. However a modular element is returned if `self` was already modular.

If `parent.has_reduce_hom() == True` then the result is reduced to be an element of the corresponding forms space if possible.

In particular this is the case if `self` is a (homogeneous) element of a forms space.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.graded_ring import␣
↪QuasiMeromorphicModularFormsRing
sage: MR = QuasiMeromorphicModularFormsRing(n=7, red_hom=True)
sage: n = MR.hecke_n()
sage: Delta = MR.Delta().full_reduce()
sage: E2 = MR.E2().full_reduce()
sage: E4 = MR.E4().full_reduce()
sage: E6 = MR.E6().full_reduce()
sage: f_rho = MR.f_rho().full_reduce()
sage: f_i = MR.f_i().full_reduce()
sage: f_inf = MR.f_inf().full_reduce()

sage: f_rho.serre_derivative() == -1/n * f_i
True
sage: f_i.serre_derivative()   == -1/2 * E4 * f_rho
True
sage: f_inf.serre_derivative() == 0
True
sage: E2.serre_derivative()    == -(n-2)/(4*n) * (E2^2 + E4)
True
sage: E4.serre_derivative()    == -(n-2)/n * E6
True
sage: E6.serre_derivative()    == -1/2 * E4^2 - (n-3)/n * E6^2 / E4
True
sage: E6.serre_derivative().parent()
ModularForms(n=7, k=8, ep=1) over Integer Ring
```

```
sage: MR = QuasiMeromorphicModularFormsRing(n=infinity, red_hom=True)
sage: Delta = MR.Delta().full_reduce()
sage: E2 = MR.E2().full_reduce()
sage: E4 = MR.E4().full_reduce()
sage: E6 = MR.E6().full_reduce()
sage: f_i = MR.f_i().full_reduce()
sage: f_inf = MR.f_inf().full_reduce()

sage: E4.serre_derivative()    == -E4 * f_i
True
sage: f_i.serre_derivative()   == -1/2 * E4
True
sage: f_inf.serre_derivative() == 0
True
sage: E2.serre_derivative()    == -1/4 * (E2^2 + E4)
True
sage: E4.serre_derivative()    == -E6
True
sage: E6.serre_derivative()    == -1/2 * E4^2 - E6^2 / E4
True
sage: E6.serre_derivative().parent()
ModularForms(n=+Infinity, k=8, ep=1) over Integer Ring
```

sqrt ( )

Try to return the square root of `self`. I.e. the element corresponding to `sqrt(self.rat())`.

Whether this works or not depends on whether `sqrt(self.rat())` works and coerces into `self.parent().rat_field()`.

Note that the parent might (probably will) change.

If `parent.has_reduce_hom() == True` then the result is reduced to be an element of the corresponding forms space if possible.

In particular this is the case if `self` is a (homogeneous) element of a forms space.

Todo

Make square root in the underlying rational field work.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.space import QuasiModularForms
sage: E2=QuasiModularForms(k=2, ep=-1).E2()
sage: sqrt(E2^2)
Traceback (most recent call last):
...
NotImplementedError: is_square() not implemented for elements of Multivariate␣
↪Polynomial Ring in x, y, z, d over Integer Ring
```

weight ( )

Return the weight of `self`.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.graded_ring import␣
↪QuasiModularFormsRing
sage: from sage.modular.modform_hecketriangle.space import ModularForms
sage: x,y,z,d = var("x,y,z,d")
sage: QuasiModularFormsRing()(x+y).weight() is None
True
sage: ModularForms(n=18).f_i().weight()
9/4
sage: ModularForms(n=infinity).f_inf().weight()
4
```

CONSTRUCTOR FOR SPACES OF MODULAR FORMS FOR HECKE TRIANGLE GROUPS BASED ON A TYPE

AUTHORS:

- Jonas Jermann (2013): initial version

`sage.modular.modform_hecketriangle.constructor.`**`FormsRing`**`(`*analytic_type*, *group=3*, *base_ring=Integer Ring*, *red_hom=False*`)`

Return the FormsRing with the given `analytic_type`, `group` `base_ring` and variable `red_hom`.

INPUT:

- **`analytic_type` – An element of `AnalyticType()` describing** the analytic type of the space.
- **`group` – The index of the (Hecke triangle) group of the space** (default: 3‘).
- **`base_ring` – The base ring of the space** (default: `ZZ`).
- **`red_hom` – The (boolean= variable `red_hom` of the space** (default: `False`).

For the variables `group`, `base_ring`, `red_hom` the same arguments as for the class `FormsRing_abstract` can be used. The variables will then be put in canonical form.

OUTPUT:

The FormsRing with the given properties.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.constructor import FormsRing
sage: FormsRing("cusp", group=5, base_ring=CC)
CuspFormsRing(n=5) over Complex Field with 53 bits of precision

sage: FormsRing("holo")
ModularFormsRing(n=3) over Integer Ring

sage: FormsRing("weak", group=6, base_ring=ZZ, red_hom=True)
WeakModularFormsRing(n=6) over Integer Ring

sage: FormsRing("mero", group=7, base_ring=ZZ)
MeromorphicModularFormsRing(n=7) over Integer Ring

sage: FormsRing(["quasi", "cusp"], group=5, base_ring=CC)
QuasiCuspFormsRing(n=5) over Complex Field with 53 bits of precision

sage: FormsRing(["quasi", "holo"])
QuasiModularFormsRing(n=3) over Integer Ring
```

```
sage: FormsRing(["quasi", "weak"], group=6, base_ring=ZZ, red_hom=True)
QuasiWeakModularFormsRing(n=6) over Integer Ring

sage: FormsRing(["quasi", "mero"], group=7, base_ring=ZZ, red_hom=True)
QuasiMeromorphicModularFormsRing(n=7) over Integer Ring

sage: FormsRing(["quasi", "cusp"], group=infinity)
QuasiCuspFormsRing(n=+Infinity) over Integer Ring
```

sage.modular.modform_hecketriangle.constructor.**FormsSpace**(*analytic_type*, *group=3*, *base_ring=Integer Ring*, *k=0*, *ep=None*)

Return the FormsSpace with the given `analytic_type`, `group` `base_ring` and degree (`k`, `ep`).

INPUT:

- **`analytic_type` – An element of `AnalyticType()` describing** the analytic type of the space.
- **`group` – The index of the (Hecke triangle) group of the** space (default: 3).
- **`base_ring` – The base ring of the space** (default: `ZZ`).
- **`k` – The weight of the space, a rational number** (default: `0`).
- **`ep` – The multiplier of the space, 1, -1** or `None` (in case `ep` should be determined from `k`). Default: `None`.

For the variables `group`, `base_ring`, `k`, `ep` the same arguments as for the class `FormsSpace_abstract` can be used. The variables will then be put in canonical form. In particular the multiplier `ep` is calculated as usual from `k` if `ep == None`.

OUTPUT:

The FormsSpace with the given properties.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.constructor import FormsSpace
sage: FormsSpace([])
ZeroForms(n=3, k=0, ep=1) over Integer Ring
sage: FormsSpace(["quasi"]) # not implemented

sage: FormsSpace("cusp", group=5, base_ring=CC, k=12, ep=1)
CuspForms(n=5, k=12, ep=1) over Complex Field with 53 bits of precision

sage: FormsSpace("holo")
ModularForms(n=3, k=0, ep=1) over Integer Ring

sage: FormsSpace("weak", group=6, base_ring=ZZ, k=0, ep=-1)
WeakModularForms(n=6, k=0, ep=-1) over Integer Ring

sage: FormsSpace("mero", group=7, base_ring=ZZ, k=2, ep=-1)
MeromorphicModularForms(n=7, k=2, ep=-1) over Integer Ring

sage: FormsSpace(["quasi", "cusp"], group=5, base_ring=CC, k=12, ep=1)
QuasiCuspForms(n=5, k=12, ep=1) over Complex Field with 53 bits of precision

sage: FormsSpace(["quasi", "holo"])
QuasiModularForms(n=3, k=0, ep=1) over Integer Ring
```

```
sage: FormsSpace(["quasi", "weak"], group=6, base_ring=ZZ, k=0, ep=-1)
QuasiWeakModularForms(n=6, k=0, ep=-1) over Integer Ring

sage: FormsSpace(["quasi", "mero"], group=7, base_ring=ZZ, k=2, ep=-1)
QuasiMeromorphicModularForms(n=7, k=2, ep=-1) over Integer Ring

sage: FormsSpace(["quasi", "cusp"], group=infinity, base_ring=ZZ, k=2, ep=-1)
QuasiCuspForms(n=+Infinity, k=2, ep=-1) over Integer Ring
```

`sage.modular.modform_hecketriangle.constructor.`**`rational_type`**(*f*, *n=3*, *base_ring=Integer Ring*)

Return the basic analytic properties that can be determined directly from the specified rational function `f` which is interpreted as a representation of an element of a FormsRing for the Hecke Triangle group with parameter `n` and the specified `base_ring`.

In particular the following degree of the generators is assumed:

$deg(1) := (0, 1)$ $deg(x) := (4/(n-2), 1)$ $deg(y) := (2n/(n-2), -1)$ $deg(z) := (2, -1)$

The meaning of homogeneous elements changes accordingly.

INPUT:

- `f` – A rational function in `x,y,z,d` over `base_ring`.
- **`n`** **– An integer greater or equal to** 3 **corresponding** to the `HeckeTriangleGroup` with that parameter (default: 3).
- **`base_ring`** **– The base ring of the corresponding forms ring, resp.** polynomial ring (default: `ZZ`).

OUTPUT:

A tuple `(elem,homo,k,ep,analytic_type)` describing the basic analytic properties of f (with the interpretation indicated above).

- `elem` – `True` if f has a homogeneous denominator.
- `homo` – `True` if f also has a homogeneous numerator.
- **`k`** **– `None` if** f **is not homogeneneous, otherwise** the weight of f (which is the first component of its degree).
- **`ep`** **– `None` if** f **is not homogeneous, otherwise** the multiplier of f (which is the second component of its degree)
- `analytic_type` – The `AnalyticType` of f.

For the zero function the degree $(0, 1)$ is choosen.

This function is (heavily) used to determine the type of elements and to check if the element really is contained in its parent.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.constructor import rational_type
sage: (x,y,z,d) = var("x,y,z,d")

sage: rational_type(0, n=4)
(True, True, 0, 1, zero)

sage: rational_type(1, n=12)
```

```
(True, True, 0, 1, modular)

sage: rational_type(x^3 - y^2)
(True, True, 12, 1, cuspidal)

sage: rational_type(x * z, n=7)
(True, True, 14/5, -1, quasi modular)

sage: rational_type(1/(x^3 - y^2) + z/d)
(True, False, None, None, quasi weakly holomorphic modular)

sage: rational_type(x^3/(x^3 - y^2))
(True, True, 0, 1, weakly holomorphic modular)

sage: rational_type(1/(x + z))
(False, False, None, None, None)

sage: rational_type(1/x + 1/z)
(True, False, None, None, quasi meromorphic modular)

sage: rational_type(d/x, n=10)
(True, True, -1/2, 1, meromorphic modular)

sage: rational_type(1.1 * z * (x^8-y^2), n=8, base_ring=CC)
(True, True, 22/3, -1, quasi cuspidal)

sage: rational_type(x-y^2, n=infinity)
(True, True, 4, 1, modular)

sage: rational_type(x*(x-y^2), n=infinity)
(True, True, 8, 1, cuspidal)

sage: rational_type(1/x, n=infinity)
(True, True, -4, 1, weakly holomorphic modular)
```

CHAPTER

SEVEN

FUNCTOR CONSTRUCTION FOR ALL SPACES

AUTHORS:

- Jonas Jermann (2013): initial version

class `sage.modular.modform_hecketriangle.functors.`**`BaseFacade`**`(`*ring*`)`

Bases: `sage.structure.parent.Parent`, `sage.structure.unique_representation.UniqueRepresenta`

BaseFacade of a ring.

This class is used to distinguish the construction of constant elements (modular forms of weight 0) over the given ring and the construction of `FormsRing` or `FormsSpace` based on the BaseFacade of the given ring.

If that distinction was not made then ring elements couldn't be considered as constant modular forms in e.g. binary operations. Instead the coercion model would assume that the ring element lies in the common parent of the ring element and e.g. a `FormsSpace` which would give the `FormsSpace` over the ring. However this is not correct, the `FormsSpace` might (and probably will) not even contain the (constant) ring element. Hence we use the `BaseFacade` to distinguish the two cases.

Since the `BaseFacade` of a ring embedds into that ring, a common base (resp. a coercion) between the two (or even a more general ring) can be found, namely the ring (not the `BaseFacade` of it).

`sage.modular.modform_hecketriangle.functors.`**`ConstantFormsSpaceFunctor`**`(`*group*`)`

Construction functor for the space of constant forms.

When determining a common parent between a ring and a forms ring or space this functor is first applied to the ring.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.functors import␣
↪(ConstantFormsSpaceFunctor, FormsSpaceFunctor)
sage: ConstantFormsSpaceFunctor(4) == FormsSpaceFunctor("holo", 4, 0, 1)
True
sage: ConstantFormsSpaceFunctor(4)
ModularFormsFunctor(n=4, k=0, ep=1)
```

class `sage.modular.modform_hecketriangle.functors.`**`FormsRingFunctor`**`(`*analytic_type*, *group*, *red_hom*`)`

Bases: `sage.categories.pushout.ConstructionFunctor`

Construction functor for forms rings.

NOTE:

When the base ring is not a `BaseFacade` the functor is first merged with the ConstantFormsSpaceFunctor. This case occurs in the pushout constructions. (when trying to find a common parent between a forms ring and a ring which is not a `BaseFacade`).

`AnalyticType`
alias of *`AnalyticType`*

`merge` (*other*)
Return the merged functor of `self` and `other` .

It is only possible to merge instances of `FormsSpaceFunctor` and `FormsRingFunctor` . Also only if they share the same group. An `FormsSubSpaceFunctors` is replaced by its ambient space functor.

The analytic type of the merged functor is the extension of the two analytic types of the functors. The `red_hom` parameter of the merged functor is the logical `and` of the two corresponding `red_hom` parameters (where a forms space is assumed to have it set to `True`).

Two `FormsSpaceFunctor` with different (k,ep) are merged to a corresponding `FormsRingFunctor` . Otherwise the corresponding (extended) `FormsSpaceFunctor` is returned.

A `FormsSpaceFunctor` and `FormsRingFunctor` are merged to a corresponding (extended) `FormsRingFunctor` .

Two `FormsRingFunctors` are merged to the corresponding (extended) `FormsRingFunctor` .

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.functors import␣
↪(FormsSpaceFunctor, FormsRingFunctor)
sage: functor1 = FormsRingFunctor("mero", group=6, red_hom=True)
sage: functor2 = FormsRingFunctor(["quasi", "cusp"], group=6, red_hom=False)
sage: functor3 = FormsSpaceFunctor("weak", group=6, k=0, ep=1)
sage: functor4 = FormsRingFunctor("mero", group=5, red_hom=True)

sage: functor1.merge(functor1) is functor1
True
sage: functor1.merge(functor4) is None
True
sage: functor1.merge(functor2)
QuasiMeromorphicModularFormsRingFunctor(n=6)
sage: functor1.merge(functor3)
MeromorphicModularFormsRingFunctor(n=6, red_hom=True)
```

class `sage.modular.modform_hecketriangle.functors.`**`FormsSpaceFunctor`** (*analytic_type*, *group*, *k*, *ep*)

Bases: `sage.categories.pushout.ConstructionFunctor`

Construction functor for forms spaces.

NOTE:

When the base ring is not a `BaseFacade` the functor is first merged with the ConstantFormsSpaceFunctor. This case occurs in the pushout constructions (when trying to find a common parent between a forms space and a ring which is not a `BaseFacade`).

`AnalyticType`
alias of *`AnalyticType`*

`merge` (*other*)
Return the merged functor of `self` and `other` .

It is only possible to merge instances of `FormsSpaceFunctor` and `FormsRingFunctor`. Also only if they share the same group. An `FormsSubSpaceFunctors` is replaced by its ambient space functor.

The analytic type of the merged functor is the extension of the two analytic types of the functors. The `red_hom` parameter of the merged functor is the logical `and` of the two corresponding `red_hom` parameters (where a forms space is assumed to have it set to `True`).

Two `FormsSpaceFunctor` with different (k,ep) are merged to a corresponding `FormsRingFunctor`. Otherwise the corresponding (extended) `FormsSpaceFunctor` is returned.

A `FormsSpaceFunctor` and `FormsRingFunctor` are merged to a corresponding (extended) `FormsRingFunctor`.

Two `FormsRingFunctors` are merged to the corresponding (extended) `FormsRingFunctor`.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.functors import␣
↪(FormsSpaceFunctor, FormsRingFunctor)
sage: functor1 = FormsSpaceFunctor("holo", group=5, k=0, ep=1)
sage: functor2 = FormsSpaceFunctor(["quasi", "cusp"], group=5, k=10/3, ep=-1)
sage: functor3 = FormsSpaceFunctor(["quasi", "mero"], group=5, k=0, ep=1)
sage: functor4 = FormsRingFunctor("cusp", group=5, red_hom=False)
sage: functor5 = FormsSpaceFunctor("holo", group=4, k=0, ep=1)

sage: functor1.merge(functor1) is functor1
True
sage: functor1.merge(functor5) is None
True
sage: functor1.merge(functor2)
QuasiModularFormsRingFunctor(n=5, red_hom=True)
sage: functor1.merge(functor3)
QuasiMeromorphicModularFormsFunctor(n=5, k=0, ep=1)
sage: functor1.merge(functor4)
ModularFormsRingFunctor(n=5)
```

class `sage.modular.modform_hecketriangle.functors.`**`FormsSubSpaceFunctor`**(*ambient_space_functor*, *generators*)

Bases: `sage.categories.pushout.ConstructionFunctor`

Construction functor for forms sub spaces.

`merge`(*other*)

Return the merged functor of `self` and `other`.

If `other` is a `FormsSubSpaceFunctor` then first the common ambient space functor is constructed by merging the two corresponding functors.

If that ambient space functor is a FormsSpaceFunctor and the generators agree the corresponding `FormsSubSpaceFunctor` is returned.

If `other` is not a `FormsSubSpaceFunctor` then `self` is merged as if it was its ambient space functor.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.functors import␣
↪(FormsSpaceFunctor, FormsSubSpaceFunctor)
sage: from sage.modular.modform_hecketriangle.space import ModularForms
```

```
sage: ambient_space = ModularForms(n=4, k=12, ep=1)
sage: ambient_space_functor1 = FormsSpaceFunctor("holo", group=4, k=12, ep=1)
sage: ambient_space_functor2 = FormsSpaceFunctor("cusp", group=4, k=12, ep=1)
sage: ss_functor1 = FormsSubSpaceFunctor(ambient_space_functor1, [ambient_
↪space.gen(0)])
sage: ss_functor2 = FormsSubSpaceFunctor(ambient_space_functor2, [ambient_
↪space.gen(0)])
sage: ss_functor3 = FormsSubSpaceFunctor(ambient_space_functor2, [2*ambient_
↪space.gen(0)])
sage: merged_ambient = ambient_space_functor1.merge(ambient_space_functor2)
sage: merged_ambient
ModularFormsFunctor(n=4, k=12, ep=1)
sage: functor4 = FormsSpaceFunctor(["quasi", "cusp"], group=4, k=10, ep=-1)

sage: ss_functor1.merge(ss_functor1) is ss_functor1
True
sage: ss_functor1.merge(ss_functor2)
FormsSubSpaceFunctor with 2 generators for the ModularFormsFunctor(n=4, k=12,␣
↪ep=1)
sage: ss_functor1.merge(ss_functor2) == FormsSubSpaceFunctor(merged_ambient,␣
↪[ambient_space.gen(0), ambient_space.gen(0)])
True
sage: ss_functor1.merge(ss_functor3) == FormsSubSpaceFunctor(merged_ambient,␣
↪[ambient_space.gen(0), 2*ambient_space.gen(0)])
True
sage: ss_functor1.merge(ambient_space_functor2) == merged_ambient
True
sage: ss_functor1.merge(functor4)
QuasiModularFormsRingFunctor(n=4, red_hom=True)
```

CHAPTER

EIGHT

HECKE TRIANGLE GROUPS

AUTHORS:

- Jonas Jermann (2013): initial version

class `sage.modular.modform_hecketriangle.hecke_triangle_groups.`**`HeckeTriangleGroup`** `(`*n*`)`

Bases: `sage.groups.matrix_gps.finitely_generated.FinitelyGeneratedMatrixGroup_generic`, `sage.structure.unique_representation.UniqueRepresentation`

Hecke triangle group (2, n, infinity).

`Element`

alias of `HeckeTriangleGroupElement`

`I` `()`

Return the identity element/matrix for `self`.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.hecke_triangle_groups import␣
↪HeckeTriangleGroup
sage: HeckeTriangleGroup(10).I()
[1 0]
[0 1]

sage: HeckeTriangleGroup(10).I().parent()
Hecke triangle group for n = 10
```

`S` `()`

Return the generator of `self` corresponding to the conformal circle inversion.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.hecke_triangle_groups import␣
↪HeckeTriangleGroup
sage: HeckeTriangleGroup(3).S()
[ 0 -1]
[ 1  0]
sage: HeckeTriangleGroup(10).S()
[ 0 -1]
[ 1  0]
sage: HeckeTriangleGroup(10).S()^2 == -HeckeTriangleGroup(10).I()
True
sage: HeckeTriangleGroup(10).S()^4 == HeckeTriangleGroup(10).I()
True

sage: HeckeTriangleGroup(10).S().parent()
Hecke triangle group for n = 10
```

T (*m=1*)

Return the element in `self` corresponding to the translation by `m*self.lam()` .

INPUT:

- `m` – An integer, default: `1` , namely the second generator of `self` .

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.hecke_triangle_groups import␣
↪HeckeTriangleGroup
sage: HeckeTriangleGroup(3).T()
[1 1]
[0 1]
sage: HeckeTriangleGroup(10).T(-4)
[     1 -4*lam]
[     0      1]
sage: HeckeTriangleGroup(10).T().parent()
Hecke triangle group for n = 10
```

U ( )

Return an alternative generator of `self` instead of `T`. `U` stabilizes `rho` and has order `2*self.n()` .

If `n=infinity` then `U` is parabolic and has infinite order, it then fixes the cusp `[-1]` .

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.hecke_triangle_groups import␣
↪HeckeTriangleGroup
sage: HeckeTriangleGroup(3).U()
[ 1 -1]
[ 1  0]
sage: HeckeTriangleGroup(3).U()^3 == -HeckeTriangleGroup(3).I()
True
sage: HeckeTriangleGroup(3).U()^6 == HeckeTriangleGroup(3).I()
True
sage: HeckeTriangleGroup(10).U()
[lam  -1]
[  1   0]
sage: HeckeTriangleGroup(10).U()^10 == -HeckeTriangleGroup(10).I()
True
sage: HeckeTriangleGroup(10).U()^20 == HeckeTriangleGroup(10).I()
True

sage: HeckeTriangleGroup(10).U().parent()
Hecke triangle group for n = 10
```

V (*j*)

Return the j'th generator for the usual representatives of conjugacy classes of `self` . It is given by `V=U^(j-1)*T` .

INPUT:

- **`j` – Any integer. To get the usual representatives** `j` should range from `1` to `self.n()-1` .

OUTPUT:

The corresponding matrix/element. The matrix is parabolic if `j` is congruent to +-1 modulo `self.n()` . It is elliptic if `j` is congruent to 0 modulo `self.n()` . It is hyperbolic otherwise.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.hecke_triangle_groups import␣
↪HeckeTriangleGroup
sage: G = HeckeTriangleGroup(3)
sage: G.V(0) == -G.S()
True
sage: G.V(1) == G.T()
True
sage: G.V(2)
[1 0]
[1 1]
sage: G.V(3) == G.S()
True

sage: G = HeckeTriangleGroup(5)
sage: G.element_repr_method("default")
sage: G.V(1)
[  1 lam]
[  0   1]
sage: G.V(2)
[lam lam]
[  1 lam]
sage: G.V(3)
[lam   1]
[lam lam]
sage: G.V(4)
[  1   0]
[lam   1]
sage: G.V(5) == G.S()
True
```

alpha ( )

Return the parameter `alpha` of `self`. This is the first parameter of the hypergeometric series used in the calculation of the Hauptmodul of `self`.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.hecke_triangle_groups import␣
↪HeckeTriangleGroup
sage: HeckeTriangleGroup(3).alpha()
1/12
sage: HeckeTriangleGroup(4).alpha()
1/8
sage: HeckeTriangleGroup(5).alpha()
3/20
sage: HeckeTriangleGroup(6).alpha()
1/6
sage: HeckeTriangleGroup(10).alpha()
1/5
sage: HeckeTriangleGroup(infinity).alpha()
1/4
```

base_field ( )

Return the base field of `self`.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.hecke_triangle_groups import␣
↪HeckeTriangleGroup
sage: HeckeTriangleGroup(n=infinity).base_field()
Rational Field
sage: HeckeTriangleGroup(n=7).base_field()
Number Field in lam with defining polynomial x^3 - x^2 - 2*x + 1
```

base_ring()

Return the base ring of `self`.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.hecke_triangle_groups import␣
↪HeckeTriangleGroup
sage: HeckeTriangleGroup(n=infinity).base_ring()
Integer Ring
sage: HeckeTriangleGroup(n=7).base_ring()
Maximal Order in Number Field in lam with defining polynomial x^3 - x^2 - 2*x␣
↪+ 1
```

beta()

Return the parameter `beta` of `self`. This is the second parameter of the hypergeometric series used in the calculation of the Hauptmodul of `self`.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.hecke_triangle_groups import␣
↪HeckeTriangleGroup
sage: HeckeTriangleGroup(3).beta()
5/12
sage: HeckeTriangleGroup(4).beta()
3/8
sage: HeckeTriangleGroup(5).beta()
7/20
sage: HeckeTriangleGroup(6).beta()
1/3
sage: HeckeTriangleGroup(10).beta()
3/10
sage: HeckeTriangleGroup(infinity).beta()
1/4
```

class_number(*D*, *primitive=True*)

Return the class number of `self` for the discriminant `D`. I.e. the number of conjugacy classes of (primitive) elements of discriminant `D`.

Note: Due to the 1-1 correspondence with hyperbolic fixed points resp. hyperbolic binary quadratic forms this also gives the class number in those cases.

INPUT:

- **`D` – An element of the base ring corresponding** to a valid discriminant.
- **`primitive` – If `True` (default) then only primitive** elements are considered.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.hecke_triangle_groups import␣
↪HeckeTriangleGroup
sage: G = HeckeTriangleGroup(n=4)
```

```
sage: G.class_number(4)
1
sage: G.class_number(4, primitive=False)
1
sage: G.class_number(14)
2
sage: G.class_number(32)
2
sage: G.class_number(32, primitive=False)
3
sage: G.class_number(68)
4
```

class_representatives (*D*, *primitive=True*)

Return a representative for each conjugacy class for the discriminant `D` (ignoring the sign).

If `primitive=True` only one representative for each fixed point is returned (ignoring sign).

INPUT:

- **`D` – An element of the base ring corresponding** to a valid discriminant.
- **`primitive` – If `True` (default) then only primitive** representatives are considered.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.hecke_triangle_groups import␣
↪HeckeTriangleGroup
sage: G = HeckeTriangleGroup(n=4)
sage: G.element_repr_method("conj")

sage: R = G.class_representatives(4)
sage: R
[[V(2)]]
sage: [v.continued_fraction()[1] for v in R]
[(2,)]

sage: R = G.class_representatives(0)
sage: R
[[V(3)]]
sage: [v.continued_fraction()[1] for v in R]
[(1, 2)]

sage: R = G.class_representatives(-4)
sage: R
[[S]]
sage: R = G.class_representatives(-4, primitive=False)
sage: R
[[S], [U^2]]

sage: R = G.class_representatives(G.lam()^2 - 4)
sage: R
[[U]]
sage: R = G.class_representatives(G.lam()^2 - 4, primitive=False)
sage: R
[[U], [U^(-1)]]

sage: R = G.class_representatives(14)
sage: R
[[V(2)*V(3)], [V(1)*V(2)]]
```

```
sage: [v.continued_fraction()[1] for v in R]
[(1, 2, 2), (3,)]

sage: R = G.class_representatives(32)
sage: R
[[V(3)^2*V(1)], [V(1)^2*V(3)]]
sage: [v.continued_fraction()[1] for v in R]
[(1, 2, 1, 3), (1, 4)]

sage: R = G.class_representatives(32, primitive=False)
sage: R
[[V(3)^2*V(1)], [V(1)^2*V(3)], [V(2)^2]]

sage: G.element_repr_method("default")
```

dvalue ( )

Return a symbolic expression (or an exact value in case n=3, 4, 6) for the transfinite diameter (or capacity) of `self` . I.e. the first nontrivial Fourier coefficient of the Hauptmodul for the Hecke triangle group in case it is normalized to `J_inv(i)=1` .

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.hecke_triangle_groups import␣
↪HeckeTriangleGroup
sage: HeckeTriangleGroup(3).dvalue()
1/1728
sage: HeckeTriangleGroup(4).dvalue()
1/256
sage: HeckeTriangleGroup(5).dvalue()
e^(2*euler_gamma - 4*pi/(sqrt(5) + 1) + psi(17/20) + psi(13/20))
sage: HeckeTriangleGroup(6).dvalue()
1/108
sage: HeckeTriangleGroup(10).dvalue()
e^(2*euler_gamma - 4*pi/sqrt(2*sqrt(5) + 10) + psi(4/5) + psi(7/10))
sage: HeckeTriangleGroup(infinity).dvalue()
1/64
```

element_repr_method (*method=None*)

Either return or set the representation method for elements of `self` .

INPUT:

- **`method` – If `method=None` (default) the current default representation** method is returned. Otherwise the default method is set to `method` . If `method` is not available a ValueError is raised. Possible methods are:

  `default` : Use the usual representation method for matrix group elements.

  `basic` : The representation is given as a word in `S` and powers of `T` .

  `conj` : The conjugacy representative of the element is represented as a word in powers of the basic blocks, together with an unspecified conjugation matrix.

  `block` : Same as `conj` but the conjugation matrix is specified as well.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.hecke_triangle_groups import␣
↪HeckeTriangleGroup
sage: G = HeckeTriangleGroup(5)
```

```
sage: G.element_repr_method()
'default'
sage: G.element_repr_method("basic")
sage: G.element_repr_method()
'basic'
```

get_FD (*z*)

Return a tuple (A,w) which determines how to map `z` to the usual (strict) fundamental domain of `self`.

INPUT:

- `z` – a complex number or an element of AlgebraicField().

OUTPUT:

A tuple `(A, w)`.

- `A` – a matrix in `self` such that `A.acton(w)==z` (if `z` is exact at least).
- `w` – a complex number or an element of AlgebraicField() (depending on the type `z`) which lies inside the (strict) fundamental domain of `self` (`self.in_FD(w)==True`) and which is equivalent to `z` (by the above property).

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.hecke_triangle_groups import␣
↪HeckeTriangleGroup
sage: G = HeckeTriangleGroup(8)
sage: z = AlgebraicField()(1+i/2)
sage: (A, w) = G.get_FD(z)
sage: A
[-lam    1]
[  -1    0]
sage: A.acton(w) == z
True

sage: from sage.modular.modform_hecketriangle.space import ModularForms
sage: z = (134.12 + 0.22*i).n()
sage: (A, w) = G.get_FD(z)
sage: A
[-73*lam^3 + 74*lam        73*lam^2 - 1]
[        -lam^2 + 1                 lam]
sage: w
0.769070776942... + 0.779859114103...*I
sage: z
134.120000000... + 0.220000000000...*I
sage: A.acton(w)
134.1200000... + 0.2200000000...*I
```

in_FD (*z*)

Returns `True` if `z` lies in the (strict) fundamental domain of `self`.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.hecke_triangle_groups import␣
↪HeckeTriangleGroup
sage: HeckeTriangleGroup(5).in_FD(CC(1.5/2 + 0.9*i))
True
sage: HeckeTriangleGroup(4).in_FD(CC(1.5/2 + 0.9*i))
False
```

is_arithmetic ( )

Return True if `self` is an arithmetic subgroup.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.hecke_triangle_groups import␣
↪HeckeTriangleGroup
sage: HeckeTriangleGroup(3).is_arithmetic()
True
sage: HeckeTriangleGroup(4).is_arithmetic()
True
sage: HeckeTriangleGroup(5).is_arithmetic()
False
sage: HeckeTriangleGroup(6).is_arithmetic()
True
sage: HeckeTriangleGroup(10).is_arithmetic()
False
sage: HeckeTriangleGroup(infinity).is_arithmetic()
True
```

is_discriminant (*D*, *primitive=True*)

Returns whether `D` is a discriminant of an element of `self`.

Note: Checking that something isn't a discriminant takes much longer than checking for valid discriminants.

INPUT:

- `D` – An element of the base ring.
- **`primitive` – If `True` (default) then only primitive** elements are considered.

OUTPUT:

`True` if `D` is a primitive discriminant (a discriminant of a primitive element) and `False` otherwise. If `primitive=False` then also non-primitive elements are considered.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.hecke_triangle_groups import␣
↪HeckeTriangleGroup
sage: G = HeckeTriangleGroup(n=4)

sage: G.is_discriminant(68)
True
sage: G.is_discriminant(196, primitive=False)     # long time
True
sage: G.is_discriminant(2)
False
```

lam ( )

Return the parameter `lambda` of `self`, where `lambda` is twice the real part of `rho`, lying between `1` (when `n=3`) and `2` (when `n=infinity`).

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.hecke_triangle_groups import␣
↪HeckeTriangleGroup
sage: HeckeTriangleGroup(3).lam()
1
sage: HeckeTriangleGroup(4).lam()
```

```
lam
sage: HeckeTriangleGroup(4).lam()^2
2
sage: HeckeTriangleGroup(6).lam()^2
3
sage: AA(HeckeTriangleGroup(10).lam())
1.9021130325903...?
sage: HeckeTriangleGroup(infinity).lam()
2
```

lam_minpoly ( )
Return the minimal polynomial of the corresponding lambda parameter of `self`.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.hecke_triangle_groups import␣
↪HeckeTriangleGroup
sage: HeckeTriangleGroup(10).lam_minpoly()
x^4 - 5*x^2 + 5
sage: HeckeTriangleGroup(17).lam_minpoly()
x^8 - x^7 - 7*x^6 + 6*x^5 + 15*x^4 - 10*x^3 - 10*x^2 + 4*x + 1
sage: HeckeTriangleGroup(infinity).lam_minpoly()
x - 2
```

list_discriminants (*D*, *primitive=True*, *hyperbolic=True*, *incomplete=False*)
Returns a list of all discriminants up to some upper bound `D`.

INPUT:

- **`D` – An element/discriminant of the base ring or** more generally an upper bound for the discriminant.
- **`primitive` – If `True` (default) then only primitive** discriminants are listed.
- **`hyperbolic` – If `True` (default) then only positive** discriminants are listed.
- **`incomplete` – If `True` (default: `False`) then all (also higher)** discriminants which were gathered so far are listed (however there might be missing discriminants inbetween).

OUTPUT:

A list of discriminants less than or equal to `D`.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.hecke_triangle_groups import␣
↪HeckeTriangleGroup
sage: G = HeckeTriangleGroup(n=4)
sage: G.list_discriminants(D=68)
[4, 12, 14, 28, 32, 46, 60, 68]
sage: G.list_discriminants(D=0, hyperbolic=False, primitive=False)
[-4, -2, 0]

sage: G = HeckeTriangleGroup(n=5)
sage: G.list_discriminants(D=20)
[4*lam, 7*lam + 6, 9*lam + 5]
sage: G.list_discriminants(D=0, hyperbolic=False, primitive=False)
[-4, -lam - 2, lam - 3, 0]
```

n ( )
Return the parameter `n` of `self`, where `pi/n` is the angle at `rho` of the corresponding basic hyperbolic

triangle with vertices `i`, `rho` and `infinity`.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.hecke_triangle_groups import␣
↪HeckeTriangleGroup
sage: HeckeTriangleGroup(10).n()
10
sage: HeckeTriangleGroup(infinity).n()
+Infinity
```

one ( )

Return the identity element/matrix for `self`.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.hecke_triangle_groups import␣
↪HeckeTriangleGroup
sage: G = HeckeTriangleGroup(10)
sage: G(1) == G.one()
True
sage: G(1)
[1 0]
[0 1]

sage: G(1).parent()
Hecke triangle group for n = 10
```

rational_period_functions (k, D)

Return a list of basic rational period functions of weight `k` for discriminant `D`. The list is expected to be a generating set for all rational period functions of the given weight and discriminant (unknown).

The method assumes that `D > 0`. Also see the element method $rational_period_function$ for more information.

•`k` – An even integer, the desired weight of the rational period functions.

•**D – An element of the base ring corresponding** to a valid discriminant.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.hecke_triangle_groups import␣
↪HeckeTriangleGroup
sage: G = HeckeTriangleGroup(n=4)
sage: G.rational_period_functions(k=4, D=12)
[(z^4 - 1)/z^4]
sage: G.rational_period_functions(k=-2, D=12)
[-z^2 + 1, 4*lam*z^2 - 4*lam]
sage: G.rational_period_functions(k=2, D=14)
[(z^2 - 1)/z^2, 1/z, (24*z^6 - 120*z^4 + 120*z^2 - 24)/(9*z^8 - 80*z^6 +␣
↪146*z^4 - 80*z^2 + 9), (24*z^6 - 120*z^4 + 120*z^2 - 24)/(9*z^8 - 80*z^6 +␣
↪146*z^4 - 80*z^2 + 9)]
sage: G.rational_period_functions(k=-4, D=14)
[-z^4 + 1, 16*z^4 - 16, -16*z^4 + 16]
```

reduced_elements (D)

Return all reduced (primitive) elements of discriminant `D`. Also see the element method `is_reduced()` for more information.

•**D – An element of the base ring corresponding** to a valid discriminant.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.hecke_triangle_groups import␣
↪HeckeTriangleGroup
sage: G = HeckeTriangleGroup(n=4)
sage: R = G.reduced_elements(D=12)
sage: R
[
[    5   -lam]   [      5 -3*lam]
[3*lam     -1],  [    lam     -1]
]
sage: [v.continued_fraction() for v in R]
[((), (1, 3)), ((), (3, 1))]
sage: R = G.reduced_elements(D=14)
sage: R
[
[ 5*lam      -3]   [ 5*lam      -7]   [4*lam     -3]   [3*lam     -1]
[     7 -2*lam],  [      3 -2*lam],  [    3   -lam],  [    1      0]
]
sage: [v.continued_fraction() for v in R]
[((), (1, 2, 2)), ((), (2, 2, 1)), ((), (2, 1, 2)), ((), (3,))]
```

rho ( )

Return the vertex `rho` of the basic hyperbolic triangle which describes `self`. `rho` has absolute value 1 and angle `pi/n`.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.hecke_triangle_groups import␣
↪HeckeTriangleGroup
sage: HeckeTriangleGroup(3).rho() == QQbar(1/2 + sqrt(3)/2*i)
True
sage: HeckeTriangleGroup(4).rho() == QQbar(sqrt(2)/2*(1 + i))
True
sage: HeckeTriangleGroup(6).rho() == QQbar(sqrt(3)/2 + 1/2*i)
True
sage: HeckeTriangleGroup(10).rho()
0.95105651629515...? + 0.30901699437494...?*I
sage: HeckeTriangleGroup(infinity).rho()
1
```

root_extension_embedding (*D*, *K=None*)

Return the correct embedding from the root extension field of the given discriminant `D` to the field `K`.

Also see the method `root_extension_embedding(K)` of `HeckeTriangleGroupElement` for more examples.

INPUT:

- **`D` – An element of the base ring of `self`** corresponding to a discriminant.
- **`K` – A field to which we want the (correct) embeddding.** If `K=None` (default) then `AlgebraicField()` is used for positive `D` and `AlgebraicRealField()` otherwise.

OUTPUT:

The corresponding embedding if it was found. Otherwise a ValueError is raised.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.hecke_triangle_groups import␣
↪HeckeTriangleGroup
sage: G = HeckeTriangleGroup(n=infinity)
sage: G.root_extension_embedding(32)
Ring morphism:
  From: Number Field in e with defining polynomial x^2 - 32
  To:   Algebraic Real Field
  Defn: e |--> 5.656854249492...?
sage: G.root_extension_embedding(-4)
Ring morphism:
  From: Number Field in e with defining polynomial x^2 + 4
  To:   Algebraic Field
  Defn: e |--> 2*I
sage: G.root_extension_embedding(4)
Ring Coercion morphism:
  From: Rational Field
  To:   Algebraic Real Field

sage: G = HeckeTriangleGroup(n=7)
sage: lam = G.lam()
sage: D = 4*lam^2 + 4*lam - 4
sage: G.root_extension_embedding(D, CC)
Relative number field morphism:
  From: Number Field in e with defining polynomial x^2 - 4*lam^2 - 4*lam + 4␣
↪over its base field
  To:   Complex Field with 53 bits of precision
  Defn: e |--> 4.02438434522...
        lam |--> 1.80193773580...
sage: D = lam^2 - 4
sage: G.root_extension_embedding(D)
Relative number field morphism:
  From: Number Field in e with defining polynomial x^2 - lam^2 + 4 over its␣
↪base field
  To:   Algebraic Field
  Defn: e |--> 0.?... + 0.867767478235...?*I
        lam |--> 1.801937735804...?
```

root_extension_field (*D*)

Return the quadratic extension field of the base field by the square root of the given discriminant `D`.

INPUT:

- **`D` – An element of the base ring of `self`** corresponding to a discriminant.

OUTPUT:

A relative (at most quadratic) extension to the base field of self in the variable `e` which corresponds to `sqrt(D)`. If the extension degree is `1` then the base field is returned.

The correct embedding is the positive resp. positive imaginary one. Unfortunately no default embedding can be specified for relative number fields yet.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.hecke_triangle_groups import␣
↪HeckeTriangleGroup
sage: G = HeckeTriangleGroup(n=infinity)
sage: G.root_extension_field(32)
Number Field in e with defining polynomial x^2 - 32
sage: G.root_extension_field(-4)
```

```
Number Field in e with defining polynomial x^2 + 4
sage: G.root_extension_field(4) == G.base_field()
True

sage: G = HeckeTriangleGroup(n=7)
sage: lam = G.lam()
sage: D = 4*lam^2 + 4*lam - 4
sage: G.root_extension_field(D)
Number Field in e with defining polynomial x^2 - 4*lam^2 - 4*lam + 4 over its␣
↪base field
sage: G.root_extension_field(4) == G.base_field()
True
sage: D = lam^2 - 4
sage: G.root_extension_field(D)
Number Field in e with defining polynomial x^2 - lam^2 + 4 over its base field
```

simple_elements (*D*)

Return all simple elements of discriminant `D` . Also see the element method `is_simple()` for more information.

- **`D` – An element of the base ring corresponding** to a valid discriminant.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.hecke_triangle_groups import␣
↪HeckeTriangleGroup
sage: G = HeckeTriangleGroup(n=4)
sage: G.simple_elements(D=12)
[
[  3 lam]  [  1 lam]
[lam   1], [lam   3]
]
sage: G.simple_elements(D=14)
[
[2*lam     1]  [  lam     1]  [2*lam     3]  [  lam     3]
[    3   lam], [    3 2*lam], [    1   lam], [    1 2*lam]
]
```

CHAPTER

NINE

HECKE TRIANGLE GROUP ELEMENTS

AUTHORS:

- Jonas Jermann (2014): initial version

class `sage.modular.modform_hecketriangle.hecke_triangle_group_element.` **`HeckeTriangleGroupElem`**

Bases: `sage.groups.matrix_gps.group_element.MatrixGroupElement_generic`

Elements of HeckeTriangleGroup.

`a` ( )

Return the upper left entry of `self`.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.hecke_triangle_groups import␣
↪HeckeTriangleGroup
sage: U = HeckeTriangleGroup(n=7).U()
sage: U.a()
lam
sage: U.a().parent()
Maximal Order in Number Field in lam with defining polynomial x^3 - x^2 - 2*x␣
↪+ 1
```

`acton` (*tau*)

Return the image of `tau` under the action of `self` by linear fractional transformations or by conjugation in case `tau` is an element of the parent of `self`.

It is possible to act on points of `HyperbolicPlane()`.

There is a 1-1 correspondence between hyperbolic fixed points and the corresponding primitive element in the stabilizer. The action in the two cases above is compatible with this correspondence.

INPUT:

- **`tau` – Either an element of `self` or any** element to which a linear fractional transformation can be applied in the usual way.

  In particular `infinity` is a possible argument and a possible return value.

  As mentioned it is also possible to use points of `HyperbolicPlane()`.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.hecke_triangle_groups import␣
↪HeckeTriangleGroup
sage: G = HeckeTriangleGroup(5)
sage: G.S().acton(1 + i/2)
2/5*I - 4/5
sage: G.S().acton(1 + i/2).parent()
Symbolic Ring
sage: G.S().acton(i + exp(-2))
-1/(e^(-2) + I)
sage: G.S().acton(i + exp(-2)).parent()
Symbolic Ring

sage: G.T().acton(infinity) == infinity
True
sage: G.U().acton(infinity)
lam
sage: G.V(2).acton(-G.lam()) == infinity
True

sage: G.V(2).acton(G.U()) == G.V(2)*G.U()*G.V(2).inverse()
True
sage: G.V(2).inverse().acton(G.U())
[  0  -1]
[  1 lam]

sage: p = HyperbolicPlane().PD().get_point(-I/2+1/8)
sage: G.V(2).acton(p)
Point in PD -((-(47*I + 161)*sqrt(5) - 47*I - 161)/(145*sqrt(5) + 94*I + 177)␣
↪+ I)/(I*(-(47*I + 161)*sqrt(5) - 47*I - 161)/(145*sqrt(5) + 94*I + 177) + 1)
sage: bool(G.V(2).acton(p).to_model('UHP').coordinates() == G.V(2).acton(p.to_
↪model('UHP').coordinates()))
True

sage: p = HyperbolicPlane().PD().get_point(I)
sage: G.U().acton(p)
Boundary point in PD 1/2*(sqrt(5) - 2*I + 1)/(-1/2*I*sqrt(5) - 1/2*I + 1)
sage: G.U().acton(p).to_model('UHP') == HyperbolicPlane().UHP().get_point(G.
↪lam())
True
sage: G.U().acton(p) == HyperbolicPlane().UHP().get_point(G.lam()).to_model(
↪'PD')
True
```

as_hyperbolic_plane_isometry (*model='UHP'*)

Return `self` as an isometry of `HyperbolicPlane()` (in the upper half plane model).

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.hecke_triangle_groups import␣
↪HeckeTriangleGroup
sage: el = HeckeTriangleGroup(7).V(4)
sage: el.as_hyperbolic_plane_isometry()
Isometry in UHP
[lam^2 - 1       lam]
[lam^2 - 1 lam^2 - 1]
sage: el.as_hyperbolic_plane_isometry().parent()
Set of Morphisms from Hyperbolic plane in the Upper Half Plane Model model to␣
↪Hyperbolic plane in the Upper Half Plane Model model in Category of␣
↪hyperbolic models of Hyperbolic plane
```

```
sage: el.as_hyperbolic_plane_isometry("KM").parent()
Set of Morphisms from Hyperbolic plane in the Klein Disk Model model to␣
↪Hyperbolic plane in the Klein Disk Model model in Category of hyperbolic␣
↪models of Hyperbolic plane
```

b ( )

Return the upper right entry of `self` .

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.hecke_triangle_groups import␣
↪HeckeTriangleGroup
sage: U = HeckeTriangleGroup(n=7).U()
sage: U.b()
-1
sage: U.b().parent()
Maximal Order in Number Field in lam with defining polynomial x^3 - x^2 - 2*x␣
↪+ 1
```

block_decomposition ( )

Return a tuple `(L,R,sgn)` such that `self = sgn * R.acton(prod(L)) = sgn * R*prod(L)*R.inverse()` .

In the parabolic and hyperbolic case the tuple entries in `L` are powers of basic block matrices: `V(j) = U^(j-1)*T = self.parent().V(j)` for `1 <= j <= n-1` . In the elliptic case the tuple entries are either `S` or `U` .

This decomposition data is (also) described by `_block_decomposition_data()` .

Warning: The case `n=infinity` is not verified at all and probably wrong!

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.hecke_triangle_groups import␣
↪HeckeTriangleGroup
sage: G = HeckeTriangleGroup(n=7)
sage: G.element_repr_method("basic")

sage: G.T().block_decomposition()
((T,), T^(-1), 1)
sage: G.V(2).acton(G.T(-3)).block_decomposition()
((-S*T^(-3)*S,), T, 1)
sage: (-G.V(2)^2).block_decomposition()
((T*S*T^2*S*T,), T*S*T, -1)

sage: el = (-G.V(2)*G.V(6)*G.V(3)*G.V(2)*G.V(6)*G.V(3))
sage: el.block_decomposition()
((-S*T^(-1)*S, T*S*T*S*T, T*S*T, -S*T^(-1)*S, T*S*T*S*T, T*S*T), T*S*T, -1)
sage: (G.U()^4*G.S()*G.V(2)).acton(el).block_decomposition()
((T*S*T, -S*T^(-1)*S, T*S*T*S*T, T*S*T, -S*T^(-1)*S, T*S*T*S*T),␣
↪T*S*T*S*T*S*T^2*S*T, -1)
sage: (G.V(1)^5*G.V(2)*G.V(3)^3).block_decomposition()
((T*S*T*S*T^2*S*T*S*T^2*S*T*S*T, T^5, T*S*T), T^6*S*T, 1)

sage: G.element_repr_method("default")
sage: (-G.I()).block_decomposition()
(
([1 0]    [1 0]  [-1  0]
[0 1],), [0 1], [ 0 -1]
)
sage: G.U().block_decomposition()
(
([lam  -1]    [1 0]  [1 0]
[  1   0],), [0 1], [0 1]
)
sage: (-G.S()).block_decomposition()
(
([ 0 -1]    [-1  0]  [-1  0]
[ 1  0],), [ 0 -1], [ 0 -1]
)
sage: (G.V(2)*G.V(3)).acton(G.U()^6).block_decomposition()
(
([  0   1]    [-2*lam^2 - 2*lam + 2 -2*lam^2 - 2*lam + 1]  [-1  0]
[ -1 lam],), [         -2*lam^2 + 1    -2*lam^2 - lam + 2], [ 0 -1]
)
sage: (G.U()^(-6)).block_decomposition()
(
([lam  -1]    [1 0]  [-1  0]
[  1   0],), [0 1], [ 0 -1]
)

sage: G = HeckeTriangleGroup(n=8)
sage: (G.U()^4).block_decomposition()
(
([     lam^2 - 1 -lam^3 + 2*lam]    [1 0]  [1 0]
[ lam^3 - 2*lam      -lam^2 + 1],), [0 1], [0 1]
)
sage: (G.U()^(-4)).block_decomposition()
(
([     lam^2 - 1 -lam^3 + 2*lam]    [1 0]  [-1  0]
[ lam^3 - 2*lam      -lam^2 + 1],), [0 1], [ 0 -1]
)
```

block_length (*primitive=False*)

Return the block length of `self`. The block length is given by the number of factors used for the decomposition of the conjugacy representative of `self` described in `primitive_representative()`. In particular the block length is invariant under conjugation.

The definition is mostly used for parabolic or hyperbolic elements: In particular it gives a lower bound for the (absolute value of) the trace and the discriminant for primitive hyperbolic elements. Namely `abs(trace) >= lambda * block_length` and `discriminant >= block_length^2 * lambda^2 -4`.

Warning: The case `n=infinity` is not verified at all and probably wrong!

INPUT:

- **`primitive`** – **If `True` then the conjugacy** representative of the primitive part is used instead, default: `False`.

OUTPUT:

An integer. For hyperbolic elements a non-negative integer. For parabolic elements a negative sign corresponds to taking the inverse. For elliptic elements a (non-trivial) integer with minimal absolute value is choosen. For +- the identity element `0` is returned.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.hecke_triangle_groups import␣
↪HeckeTriangleGroup
sage: G = HeckeTriangleGroup(n=7)
sage: G.T().block_length()
1
sage: G.V(2).acton(G.T(-3)).block_length()
3
sage: G.V(2).acton(G.T(-3)).block_length(primitive=True)
1
sage: (-G.V(2)).block_length()
1

sage: el = -G.V(2)^3*G.V(6)^2*G.V(3)
sage: t = el.block_length()
sage: D = el.discriminant()
sage: trace = el.trace()
sage: (trace, D, t)
(-124*lam^2 - 103*lam + 68, 65417*lam^2 + 52456*lam - 36300, 6)
sage: abs(AA(trace)) >= AA(G.lam()*t)
True
sage: AA(D) >= AA(t^2 * G.lam() - 4)
True
sage: (el^3).block_length(primitive=True) == t
True

sage: el = (G.U()^4*G.S()*G.V(2)).acton(-G.V(2)^3*G.V(6)^2*G.V(3))
sage: t = el.block_length()
sage: D = el.discriminant()
sage: trace = el.trace()
sage: (trace, D, t)
(-124*lam^2 - 103*lam + 68, 65417*lam^2 + 52456*lam - 36300, 6)
sage: abs(AA(trace)) >= AA(G.lam()*t)
True
sage: AA(D) >= AA(t^2 * G.lam() - 4)
True
sage: (el^(-2)).block_length(primitive=True) == t
True

sage: el = G.V(1)^5*G.V(2)*G.V(3)^3
sage: t = el.block_length()
sage: D = el.discriminant()
sage: trace = el.trace()
sage: (trace, D, t)
(284*lam^2 + 224*lam - 156, 330768*lam^2 + 265232*lam - 183556, 9)
sage: abs(AA(trace)) >= AA(G.lam()*t)
True
sage: AA(D) >= AA(t^2 * G.lam() - 4)
True
sage: (el^(-1)).block_length(primitive=True) == t
True

sage: (G.V(2)*G.V(3)).acton(G.U()^6).block_length()
1
sage: (G.V(2)*G.V(3)).acton(G.U()^6).block_length(primitive=True)
1

sage: (-G.I()).block_length()
0
```

```
sage: G.U().block_length()
1
sage: (-G.S()).block_length()
1
```

c ()

Return the lower left entry of `self`.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.hecke_triangle_groups import␣
↪HeckeTriangleGroup
sage: U = HeckeTriangleGroup(n=7).U()
sage: U.c()
1
```

conjugacy_type (*ignore_sign=True*, *primitive=False*)

Return a unique description of the conjugacy class of `self` (by default only up to a sign).

Warning: The case `n=infinity` is not verified at all and probably wrong!

INPUT:

- **`ignore_sign` – If `True` (default) then the conjugacy** classes are only considered up to a sign.
- **`primitive` – If `True` then the conjugacy class of** the primitive part is considered instead and the sign is ignored, default: `False`.

OUTPUT:

A unique representative for the given block data (without the conjugation matrix) among all cyclic permutations. If `ignore_sign=True` then the sign is excluded as well.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.hecke_triangle_groups import␣
↪HeckeTriangleGroup
sage: G = HeckeTriangleGroup(n=7)
sage: (-G.I()).conjugacy_type()
((6, 0),)
sage: G.U().acton(G.S()).conjugacy_type()
(0, 1)
sage: (G.U()^4).conjugacy_type()
(1, -3)
sage: ((G.V(2)*G.V(3)^2*G.V(2)*G.V(3))^2).conjugacy_type()
((3, 2), (2, 1), (3, 1), (2, 1), (3, 2), (2, 1), (3, 1), (2, 1))

sage: (-G.I()).conjugacy_type(ignore_sign=False)
(((6, 0),), -1)
sage: G.S().conjugacy_type(ignore_sign=False)
((0, 1), 1)
sage: (G.U()^4).conjugacy_type(ignore_sign=False)
((1, -3), -1)
sage: G.U().acton((G.V(2)*G.V(3)^2*G.V(2)*G.V(3))^2).conjugacy_type(ignore_
↪sign=False)
(((3, 2), (2, 1), (3, 1), (2, 1), (3, 2), (2, 1), (3, 1), (2, 1)), 1)

sage: (-G.I()).conjugacy_type(primitive=True)
((6, 0),)
sage: G.S().conjugacy_type(primitive=True)
```

```
(0, 1)
sage: G.V(2).acton(G.U()^4).conjugacy_type(primitive=True)
(1, 1)
sage: (G.V(3)^2).conjugacy_type(primitive=True)
((3, 1),)
sage: G.S().acton((G.V(2)*G.V(3)^2*G.V(2)*G.V(3))^2).conjugacy_
↪type(primitive=True)
((3, 2), (2, 1), (3, 1), (2, 1))
```

continued_fraction ()

For hyperbolic and parabolic elements: Return the (negative) lambda-continued fraction expansion (lambda-CF) of the (attracting) hyperbolic fixed point of `self` .

Let `r_j in Z` for `j >= 0` . A finite lambda-CF is defined as: `[r_0; r_1,...,r_k] := (T^(r_0)*S* ... *T^(r_k)*S)(infinity)` , where `S` and `T` are the generators of `self` . An infinite lambda-CF is defined as a corresponding limit value (k->infinity) if it exists.

In this case the lambda-CF of parabolic and hyperbolic fixed points are returned which have an eventually periodic lambda-CF. The parabolic elements are exactly those with a cyclic permutation of the period `[2,1,...,1]` with `n-3` ones.

Warning: The case `n=infinity` is not verified at all and probably wrong!

OUTPUT:

A tuple `(preperiod,period)` with the preperiod and period tuples of the lambda-CF.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.hecke_triangle_groups import␣
↪HeckeTriangleGroup
sage: G = HeckeTriangleGroup(n=7)
sage: G.T().continued_fraction()
((0, 1), (1, 1, 1, 1, 2))
sage: G.V(2).acton(G.T(-3)).continued_fraction()
((), (2, 1, 1, 1, 1))
sage: (-G.V(2)).continued_fraction()
((1,), (2,))
sage: (-G.V(2)^3*G.V(6)^2*G.V(3)).continued_fraction()
((1,), (2, 2, 2, 1, 1, 1, 1, 2, 1, 1, 1, 1, 2, 1, 2))
sage: (G.U()^4*G.S()*G.V(2)).acton(-G.V(2)^3*G.V(6)^2*G.V(3)).continued_
↪fraction()
((1, 1, 1, 2), (2, 2, 2, 2, 1, 1, 1, 1, 2, 1, 1, 1, 1, 2, 1))
sage: (G.V(1)^5*G.V(2)*G.V(3)^3).continued_fraction()
((6,), (2, 1, 2, 1, 2, 1, 7))

sage: G = HeckeTriangleGroup(n=8)
sage: G.T().continued_fraction()
((0, 1), (1, 1, 1, 1, 1, 2))
sage: G.V(2).acton(G.T(-3)).continued_fraction()
((), (2, 1, 1, 1, 1, 1))
sage: (-G.V(2)).continued_fraction()
((1,), (2,))
sage: (-G.V(2)^3*G.V(6)^2*G.V(3)).continued_fraction()
((1,), (2, 2, 2, 1, 1, 1, 1, 2, 1, 1, 1, 1, 2, 1, 2))
sage: (G.U()^4*G.S()*G.V(2)).acton(-G.V(2)^3*G.V(6)^2*G.V(3)).continued_
↪fraction()
((1, 1, 1, 2), (2, 2, 2, 2, 1, 1, 1, 1, 2, 1, 1, 1, 1, 2, 1))
sage: (G.V(1)^5*G.V(2)*G.V(3)^3).continued_fraction()
((6,), (2, 1, 2, 1, 2, 1, 7))
```

```
sage: (G.V(2)^3*G.V(5)*G.V(1)*G.V(6)^2*G.V(4)).continued_fraction()
((1,), (2, 2, 2, 1, 1, 1, 3, 1, 1, 1, 1, 2, 1, 1, 1, 1, 2, 1, 1, 2))
```

d ( )

Return the lower right of `self`.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.hecke_triangle_groups import␣
↪HeckeTriangleGroup
sage: U = HeckeTriangleGroup(n=7).U()
sage: U.d()
0
```

discriminant ( )

Return the discriminant of `self` which corresponds to the discriminant of the corresponding quadratic form of `self`.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.hecke_triangle_groups import␣
↪HeckeTriangleGroup
sage: G = HeckeTriangleGroup(n=7)
sage: G.V(3).discriminant()
4*lam^2 + 4*lam - 4
sage: AA(G.V(3).discriminant())
16.19566935808922?
```

fixed_points (*embedded=False*, *order='default'*)

Return a pair of (mutually conjugate) fixed points of `self` in a possible quadratic extension of the base field.

INPUT:

- **embedded – If `True` the fixed points are embedded into** `AlgebraicRealField` resp. `AlgebraicField`. Default: `False`.
- **order – If `order="none"` the fixed points are choosen** and ordered according to a fixed formula.

  If `order="sign"` the fixed points are always ordered according to the sign in front of the square root.

  If `order="default"` (default) then in case the fixed points are hyperbolic they are ordered according to the sign of the trace of `self` instead, such that the attracting fixed point comes first.

  If `order="trace"` the fixed points are always ordered according to the sign of the trace of `self`. If the trace is zero they are ordered by the sign in front of the square root. In particular the fixed_points in this case remain the same for `-self`.

OUTPUT:

If `embedded=True` an element of either `AlgebraicRealField` or `AlgebraicField` is returned. Otherwise an element of a relative field extension over the base field of (the parent of) `self` is returned.

Warning: Relative field extensions don't support default embeddings. So the correct embedding (which is the positive resp. imaginary positive one) has to be choosen.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.hecke_triangle_groups import␣
↪HeckeTriangleGroup
sage: G = HeckeTriangleGroup(n=infinity)
sage: (-G.T(-4)).fixed_points()
(+Infinity, +Infinity)
sage: (-G.S()).fixed_points()
(1/2*e, -1/2*e)
sage: p = (-G.S()).fixed_points(embedded=True)[0]
sage: p
I
sage: (-G.S()).acton(p) == p
True
sage: (-G.V(2)).fixed_points()
(1/2*e, -1/2*e)
sage: (-G.V(2)).fixed_points() == G.V(2).fixed_points()
True
sage: p = (-G.V(2)).fixed_points(embedded=True)[1]
sage: p
-1.732050807568878?
sage: (-G.V(2)).acton(p) == p
True

sage: G = HeckeTriangleGroup(n=7)
sage: (-G.S()).fixed_points()
(1/2*e, -1/2*e)
sage: p = (-G.S()).fixed_points(embedded=True)[1]
sage: p
-I
sage: (-G.S()).acton(p) == p
True
sage: (G.U()^4).fixed_points()
((1/2*lam^2 - 1/2*lam - 1/2)*e + 1/2*lam, (-1/2*lam^2 + 1/2*lam + 1/2)*e + 1/
↪2*lam)
sage: pts = (G.U()^4).fixed_points(order="trace")
sage: (G.U()^4).fixed_points() == [pts[1], pts[0]]
False
sage: (G.U()^4).fixed_points(order="trace") == (-G.U()^4).fixed_points(order=
↪"trace")
True
sage: (G.U()^4).fixed_points() == (G.U()^4).fixed_points(order="none")
True
sage: (-G.U()^4).fixed_points() == (G.U()^4).fixed_points()
True
sage: (-G.U()^4).fixed_points(order="none") == pts
True
sage: p = (G.U()^4).fixed_points(embedded=True)[1]
sage: p
0.9009688679024191? - 0.4338837391175581?*I
sage: (G.U()^4).acton(p) == p
True
sage: (-G.V(5)).fixed_points()
((1/2*lam^2 - 1/2*lam - 1/2)*e, (-1/2*lam^2 + 1/2*lam + 1/2)*e)
sage: (-G.V(5)).fixed_points() == G.V(5).fixed_points()
True
sage: p = (-G.V(5)).fixed_points(embedded=True)[0]
sage: p
0.6671145837954892?
sage: (-G.V(5)).acton(p) == p
```

```
True
```

is_elliptic ( )

Return whether `self` is an elliptic matrix.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.hecke_triangle_groups import␣
↪HeckeTriangleGroup
sage: G = HeckeTriangleGroup(n=7)
sage: [ G.V(k).is_elliptic() for k in range(1,8) ]
[False, False, False, False, False, False, True]
sage: G.U().is_elliptic()
True
```

is_hecke_symmetric ( )

Return whether the conjugacy class of the primitive part of `self`, denoted by `[gamma]` is $Hecke-symmetric$: I.e. if `[gamma] == [gamma^(-1)]`.

This is equivalent to `self.simple_fixed_point_set()` beeing equal with it's $Hecke-conjugated$ set (where each fixed point is replaced by the other ($Hecke-conjugated$) fixed point.

It is also equivalent to `[Q] == [-Q]` for the corresponding hyperbolic binary quadratic form `Q`.

The method assumes that `self` is hyperbolic.

Warning: The case `n=infinity` is not verified at all and probably wrong!

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.hecke_triangle_groups import␣
↪HeckeTriangleGroup
sage: G = HeckeTriangleGroup(n=5)

sage: el = G.V(2)
sage: el.is_hecke_symmetric()
False
sage: (el.simple_fixed_point_set(), el.inverse().simple_fixed_point_set())
({1/2*e, (-1/2*lam + 1/2)*e}, {(1/2*lam - 1/2)*e, -1/2*e})

sage: el = G.V(3)*G.V(2)^(-1)*G.V(1)*G.V(6)
sage: el.is_hecke_symmetric()
False
sage: el.simple_fixed_point_set() == el.inverse().simple_fixed_point_set()
False

sage: el = G.V(2)*G.V(3)
sage: el.is_hecke_symmetric()
True
sage: el.simple_fixed_point_set()
{(lam - 3/2)*e + 1/2*lam - 1, (-lam + 3/2)*e + 1/2*lam - 1, (lam - 3/2)*e - 1/
↪2*lam + 1, (-lam + 3/2)*e - 1/2*lam + 1}
sage: el.simple_fixed_point_set() == el.inverse().simple_fixed_point_set()
True
```

is_hyperbolic ( )

Return whether `self` is a hyperbolic matrix.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.hecke_triangle_groups import␣
↪HeckeTriangleGroup
sage: G = HeckeTriangleGroup(n=7)
sage: [ G.V(k).is_hyperbolic() for k in range(1,8) ]
[False, True, True, True, True, False, False]
sage: G.U().is_hyperbolic()
False
```

is_identity ( )

Return whether `self` is the identity or minus the identity.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.hecke_triangle_groups import␣
↪HeckeTriangleGroup
sage: G = HeckeTriangleGroup(n=7)
sage: [ G.V(k).is_identity() for k in range(1,8) ]
[False, False, False, False, False, False, False]
sage: G.U().is_identity()
False
```

is_parabolic (*exclude_one=False*)

Return whether `self` is a parabolic matrix.

If `exclude_one` is set, then +- the identity element is not considered parabolic.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.hecke_triangle_groups import␣
↪HeckeTriangleGroup
sage: G = HeckeTriangleGroup(n=7)
sage: [ G.V(k).is_parabolic() for k in range(1,8) ]
[True, False, False, False, False, True, False]
sage: G.U().is_parabolic()
False
sage: G.V(6).is_parabolic(exclude_one=True)
True
sage: G.V(7).is_parabolic(exclude_one=True)
False
```

is_primitive ( )

Returns whether `self` is primitive. We call an element primitive if (up to a sign and taking inverses) it generates the full stabilizer subgroup of the corresponding fixed point. In the non-elliptic case this means that primitive elements cannot be written as a $non-trivial$ power of another element.

The notion is mostly used for hyperbolic and parabolic elements.

Warning: The case `n=infinity` is not verified at all and probably wrong!

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.hecke_triangle_groups import␣
↪HeckeTriangleGroup
sage: G = HeckeTriangleGroup(n=7)
sage: G.V(2).acton(G.T(-1)).is_primitive()
True
sage: G.T(3).is_primitive()
False
sage: (-G.V(2)^2).is_primitive()
False
```

```
sage: (G.V(1)^5*G.V(2)*G.V(3)^3).is_primitive()
True

sage: (-G.I()).is_primitive()
True
sage: (-G.U()).is_primitive()
True
sage: (-G.S()).is_primitive()
True
sage: (G.U()^6).is_primitive()
True

sage: G = HeckeTriangleGroup(n=8)
sage: (G.U()^2).is_primitive()
False
sage: (G.U()^(-4)).is_primitive()
False
sage: (G.U()^(-3)).is_primitive()
True
```

is_reduced (*require_primitive=True*, *require_hyperbolic=True*)

Returns whether `self` is reduced. We call an element reduced if the associated lambda-CF is purely periodic.

I.e. (in the hyperbolic case) if the associated hyperbolic fixed point (resp. the associated hyperbolic binary quadratic form) is reduced.

Note that if `self` is reduced then the element corresponding to the cyclic permutation of the lambda-CF (which is conjugate to the original element) is again reduced. In particular the reduced elements in the conjugacy class of `self` form a finite cycle.

Elliptic elements and +- identity are not considered reduced.

Warning: The case `n=infinity` is not verified at all and probably wrong!

INPUT:

- **`require_primitive`** – **If `True` (default) then non-primitive elements** are not considered reduced.
- **`require_hyperbolic`** – **If `True` (default) then non-hyperbolic elements** are not considered reduced.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.hecke_triangle_groups import␣
↪HeckeTriangleGroup
sage: G = HeckeTriangleGroup(n=7)
sage: G.I().is_reduced(require_hyperbolic=False)
False
sage: G.U().reduce().is_reduced(require_hyperbolic=False)
False
sage: G.T().reduce().is_reduced()
False
sage: G.T().reduce().is_reduced(require_hyperbolic=False)
True
sage: (G.V(5)^2).reduce(primitive=False).is_reduced()
False
sage: (G.V(5)^2).reduce(primitive=False).is_reduced(require_primitive=False)
True
```

```
sage: G.V(5).reduce().is_reduced()
True
sage: (-G.V(2)).reduce().is_reduced()
True
sage: (-G.V(2)^3*G.V(6)^2*G.V(3)).reduce().is_reduced()
True
sage: (G.U()^4*G.S()*G.V(2)).acton(-G.V(2)^3*G.V(6)^2*G.V(3)).reduce().is_
↪reduced()
True
```

is_reflection ( )

Return whether `self` is the usual reflection on the unit circle.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.hecke_triangle_groups import␣
↪HeckeTriangleGroup
sage: (-HeckeTriangleGroup(n=7).S()).is_reflection()
True
sage: HeckeTriangleGroup(n=7).U().is_reflection()
False
```

is_simple ( )

Return whether `self` is simple. We call an element simple if it is hyperbolic, primitive, has positive sign and if the associated hyperbolic fixed points satisfy: `alpha' < 0 < alpha` where `alpha` is the attracting fixed point for the element.

I.e. if the associated hyperbolic fixed point (resp. the associated hyperbolic binary quadratic form) is simple.

There are only finitely many simple elements for a given discriminant. They can be used to provide explicit descriptions of rational period functions.

Warning: The case `n=infinity` is not verified at all and probably wrong!

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.hecke_triangle_groups import␣
↪HeckeTriangleGroup
sage: G = HeckeTriangleGroup(n=5)

sage: el = G.V(2)
sage: el.is_simple()
True
sage: R = el.simple_elements()
sage: [v.is_simple() for v in R]
[True]
sage: (fp1, fp2) = R[0].fixed_points(embedded=True)
sage: (fp1, fp2)
(1.272019649514069?, -1.272019649514069?)
sage: fp2 < 0 < fp1
True

sage: el = G.V(3)*G.V(2)^(-1)*G.V(1)*G.V(6)
sage: el.is_simple()
False
sage: R = el.simple_elements()
sage: [v.is_simple() for v in R]
[True, True]
```

```
sage: (fp1, fp2) = R[1].fixed_points(embedded=True)
sage: fp2 < 0 < fp1
True

sage: el = G.V(1)^2*G.V(2)*G.V(4)
sage: el.is_simple()
True
sage: R = el.simple_elements()
sage: el in R
True
sage: [v.is_simple() for v in R]
[True, True, True, True]
sage: (fp1, fp2) = R[2].fixed_points(embedded=True)
sage: fp2 < 0 < fp1
True
```

is_translation (*exclude_one=False*)

Return whether `self` is a translation. If `exclude_one = True`, then the identity map is not considered as a translation.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.hecke_triangle_groups import␣
↪HeckeTriangleGroup
sage: (-HeckeTriangleGroup(n=7).T(-4)).is_translation()
True
sage: (-HeckeTriangleGroup(n=7).I()).is_translation()
True
sage: (-HeckeTriangleGroup(n=7).I()).is_translation(exclude_one=True)
False
```

linking_number ()

Let `g` denote a holomorphic primitive of `E2` in the sense: `lambda/(2*pi*i) d/dz g = E2`. Let `gamma=self` and let `M_gamma(z)` be `Log((c*z+d) * sgn(a+d))` if `c, a+d > 0`, resp. `Log((c*z+d) / i*sgn(c))` if `a+d = 0, c!=0`, resp. `0` if `c=0`. Let `k=4 * n / (n-2)`, then:

`g(gamma.acton(z) -g(z) -k*M_gamma(z)` is equal to `2*pi*i / (n-2) * self.linking_number()`.

In particular it is independent of `z` and a conjugacy invariant.

If `self` is hyperbolic then in the classical case `n=3` this is the linking number of the closed geodesic (corresponding to `self`) with the trefoil knot.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.hecke_triangle_groups import␣
↪HeckeTriangleGroup
sage: from sage.modular.modform_hecketriangle.space import QuasiModularForms

sage: def E2_primitive(z, n=3, prec=10, num_prec=53):
....:     G = HeckeTriangleGroup(n=n)
....:     MF = QuasiModularForms(group=G, k=2, ep=-1)
....:     q = MF.get_q(prec=prec)
....:     int_series = integrate((MF.E2().q_expansion(prec=prec) - 1) / q)
....:
....:     t_const = (2*pi*i/G.lam()).n(num_prec)
....:     d = MF.get_d(fix_d=True, d_num_prec=num_prec)
```

```
....:     q = exp(t_const * z)
....:     return t_const*z + sum([(int_series.coefficients()[m]).subs(d=d) *␣
↪q**int_series.exponents()[m] for m in range(len(int_series.
↪coefficients()))])

sage: def M(gamma, z, num_prec=53):
....:     a = ComplexField(num_prec)(gamma.a())
....:     b = ComplexField(num_prec)(gamma.b())
....:     c = ComplexField(num_prec)(gamma.c())
....:     d = ComplexField(num_prec)(gamma.d())
....:
....:     if c == 0:
....:         return 0
....:     elif a + d == 0:
....:         return log(-i.n(num_prec)*(c*z + d)*sign(c))
....:     else:
....:         return log((c*z+d)*sign(a+d))

sage: def num_linking_number(A, z, n=3, prec=10, num_prec=53):
....:     z = z.n(num_prec)
....:     k = 4 * n / (n - 2)
....:     return (n-2) / (2*pi*i).n(num_prec) * (E2_primitive(A.acton(z),␣
↪n=n, prec=prec, num_prec=num_prec) - E2_primitive(z, n=n, prec=prec, num_
↪prec=num_prec) - k*M(A, z, num_prec=num_prec))

sage: G = HeckeTriangleGroup(8)
sage: z = i
sage: for A in [G.S(), G.T(), G.U(), G.U()^(G.n()//2), G.U()^(-3)]:
....:     print("A={}: ".format(A.string_repr("conj")))
....:     num_linking_number(A, z, G.n())
....:     A.linking_number()
A=[S]:
0.000000000000...
0
A=[V(1)]:
6.000000000000...
6
A=[U]:
-2.00000000000...
-2
A=[U^4]:
0.596987639289... + 0.926018962976...*I
0
A=[U^(-3)]:
5.40301236071... + 0.926018962976...*I
6

sage: z = - 2.3 + 3.1*i
sage: B = G.I()
sage: for A in [G.S(), G.T(), G.U(), G.U()^(G.n()//2), G.U()^(-3)]:
....:     print("A={}: ".format(A.string_repr("conj")))
....:     num_linking_number(B.acton(A), z, G.n(), prec=100, num_prec=1000).
↪n(53)
....:     B.acton(A).linking_number()
A=[S]:
6.63923483989...e-31 + 2.45195568651...e-30*I
0
A=[V(1)]:
```

```
6.000000000000...
6
A=[U]:
-2.00000000000... + 2.45195568651...e-30*I
-2
A=[U^4]:
0.00772492873864... + 0.00668936643212...*I
0
A=[U^(-3)]:
5.99730551444... + 0.000847636355069...*I
6

sage: z = - 2.3 + 3.1*i
sage: B = G.U()
sage: for A in [G.S(), G.T(), G.U(), G.U()^(G.n()//2), G.U()^(-3)]:    # long␣
↪time
....:     print("A={}: ".format(A.string_repr("conj")))
....:     num_linking_number(B.acton(A), z, G.n(), prec=200, num_prec=5000).
↪n(53)
....:     B.acton(A).linking_number()
A=[S]:
-7.90944791339...e-34 - 9.38956758807...e-34*I
0
A=[V(1)]:
5.99999997397... - 5.96520311160...e-8*I
6
A=[U]:
-2.00000000000... - 1.33113963568...e-61*I
-2
A=[U^4]:
-2.32704571946...e-6 + 5.91899385948...e-7*I
0
A=[U^(-3)]:
6.00000032148... - 1.82676936467...e-7*I
6

sage: A = G.V(2)*G.V(3)
sage: B = G.I()
sage: num_linking_number(B.acton(A), z, G.n(), prec=200, num_prec=5000).n(53)␣
↪    # long time
6.00498424588... - 0.00702329345176...*I
sage: A.linking_number()
6

The numerical properties for anything larger are basically
too bad to make nice further tests...
```

primitive_part (*method='cf'*)

Return the primitive part of `self`. I.e. a group element `A` with non-negative trace such that `self = sign * A^power`, where `sign = self.sign()` is +- the identity (to correct the sign) and `power = self.primitive_power()`.

The primitive part itself is choosen such that it cannot be written as a non-trivial power of another element. It is a generator of the stabilizer of the corresponding (attracting) fixed point.

If `self` is elliptic then the primitive part is chosen as a conjugate of `S` or `U`.

Warning: The case `n=infinity` is not verified at all and probably wrong!

INPUT:

- **method – The method used to determine the primitive** part (see *primitive_representative()*), default: "cf". The parameter is ignored for elliptic elements or +- the identity.

  The result should not depend on the method.

OUTPUT:

The primitive part as a group element of `self` .

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.hecke_triangle_groups import␣
↪HeckeTriangleGroup
sage: G = HeckeTriangleGroup(n=7)
sage: G.element_repr_method("block")
sage: G.T().primitive_part()
(T^(-1)*S) * (V(6)) * (T^(-1)*S)^(-1)
sage: G.V(2).acton(G.T(-3)).primitive_part()
(T) * (V(6)) * (T)^(-1)
sage: (-G.V(2)).primitive_part()
(T*S*T) * (V(2)) * (T*S*T)^(-1)
sage: (-G.V(2)^3*G.V(6)^2*G.V(3)).primitive_part()
V(2)^3*V(6)^2*V(3)
sage: (G.U()^4*G.S()*G.V(2)).acton(-G.V(2)^3*G.V(6)^2*G.V(3)).primitive_part()
(T*S*T*S*T*S*T^2*S*T) * (V(2)^3*V(6)^2*V(3)) * (T*S*T*S*T*S*T^2*S*T)^(-1)
sage: (G.V(1)^5*G.V(2)*G.V(3)^3).primitive_part()
(T^6*S*T) * (V(3)^3*V(1)^5*V(2)) * (T^6*S*T)^(-1)
sage: (G.V(2)*G.V(3)).acton(G.U()^6).primitive_part()
(-T*S*T^2*S*T*S*T) * (U) * (-T*S*T^2*S*T*S*T)^(-1)

sage: (-G.I()).primitive_part()
1

sage: G.U().primitive_part()
U
sage: (-G.S()).primitive_part()
S
sage: el = (G.V(2)*G.V(3)).acton(G.U()^6)
sage: el.primitive_part()
(-T*S*T^2*S*T*S*T) * (U) * (-T*S*T^2*S*T*S*T)^(-1)
sage: el.primitive_part() == el.primitive_part(method="block")
True

sage: G.T().primitive_part()
(T^(-1)*S) * (V(6)) * (T^(-1)*S)^(-1)
sage: G.T().primitive_part(method="block")
(T^(-1)) * (V(1)) * (T^(-1))^(-1)
sage: G.V(2).acton(G.T(-3)).primitive_part() == G.V(2).acton(G.T(-3)).
↪primitive_part(method="block")
True
sage: (-G.V(2)).primitive_part() == (-G.V(2)).primitive_part(method="block")
True
sage: el = -G.V(2)^3*G.V(6)^2*G.V(3)
sage: el.primitive_part() == el.primitive_part(method="block")
True
sage: el = (G.U()^4*G.S()*G.V(2)).acton(-G.V(2)^3*G.V(6)^2*G.V(3))
sage: el.primitive_part() == el.primitive_part(method="block")
```

```
True
sage: el=G.V(1)^5*G.V(2)*G.V(3)^3
sage: el.primitive_part() == el.primitive_part(method="block")
True

sage: G.element_repr_method("default")
```

primitive_power (*method='cf'*)

Return the primitive power of `self` . I.e. an integer `power` such that `self = sign * primitive_part^power` , where `sign = self.sign()` and `primitive_part = self.primitive_part(method)` .

Warning: For the parabolic case the sign depends on the method: The "cf" method may return a negative power but the "block" method never will.

Warning: The case `n=infinity` is not verified at all and probably wrong!

INPUT:

- **`method` – The method used to determine the primitive** power (see *`primitive_representative()`*), default: "cf". The parameter is ignored for elliptic elements or +- the identity.

OUTPUT:

An integer. For +- the identity element `0` is returned, for parabolic and hyperbolic elements a positive integer. And for elliptic elements a (non-zero) integer with minimal absolute value such that `primitive_part^power` still has a positive sign.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.hecke_triangle_groups import␣
↪HeckeTriangleGroup
sage: G = HeckeTriangleGroup(n=7)
sage: G.T().primitive_power()
-1
sage: G.V(2).acton(G.T(-3)).primitive_power()
3
sage: (-G.V(2)^2).primitive_power()
2
sage: el = (-G.V(2)*G.V(6)*G.V(3)*G.V(2)*G.V(6)*G.V(3))
sage: el.primitive_power()
2
sage: (G.U()^4*G.S()*G.V(2)).acton(el).primitive_power()
2
sage: (G.V(2)*G.V(3)).acton(G.U()^6).primitive_power()
-1
sage: G.V(2).acton(G.T(-3)).primitive_power() == G.V(2).acton(G.T(-3)).
↪primitive_power(method="block")
True

sage: (-G.I()).primitive_power()
0
sage: G.U().primitive_power()
1
sage: (-G.S()).primitive_power()
1
sage: el = (G.V(2)*G.V(3)).acton(G.U()^6)
sage: el.primitive_power()
-1
```

```
sage: el.primitive_power() == (-el).primitive_power()
True
sage: (G.U()^(-6)).primitive_power()
1

sage: G = HeckeTriangleGroup(n=8)
sage: (G.U()^4).primitive_power()
4
sage: (G.U()^(-4)).primitive_power()
4
```

primitive_representative (*method='block'*)

Return a tuple (P,R) which gives the decomposition of the primitive part of self , namely R*P*R.inverse() into a specific representative P and the corresponding conjugation matrix R (the result depends on the method used).

Together they describe the primitive part of self. I.e. an element which is equal to self up to a sign after taking the appropriate power.

See _primitive_block_decomposition_data() for a description about the representative in case the default method block is used. Also see *primitive_part()* to construct the primitive part of self .

Warning: The case n=infinity is not verified at all and probably wrong!

INPUT:

- **method – block (default) or cf . The method** used to determine P and R . If self is elliptic this parameter is ignored and if self is +- the identity then the block method is used.

  With block the decomposition described in _primitive_block_decomposition_data() is used.

  With cf a reduced representative from the lambda-CF of self is used (see *continued_fraction()*). In that case P corresponds to the period and R to the preperiod.

OUTPUT:

A tuple (P,R) of group elements such that R*P*R.inverse() is a/the primitive part of self

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.hecke_triangle_groups import␣
↪HeckeTriangleGroup
sage: G = HeckeTriangleGroup(n=7)
sage: G.element_repr_method("basic")
sage: el = G.T().primitive_representative(method="cf")
sage: el
(S*T^(-1)*S*T^(-1)*S*T*S, S*T*S)
sage: (el[0]).is_primitive()
True
sage: el = G.V(2).acton(G.T(-3)).primitive_representative(method="cf")
sage: el
(-T*S*T^(-1)*S*T^(-1), 1)
sage: (el[0]).is_primitive()
True
sage: el = (-G.V(2)).primitive_representative(method="cf")
sage: el
(T^2*S, T*S)
sage: (el[0]).is_primitive()
```

```
True
sage: el = (-G.V(2)^3*G.V(6)^2*G.V(3)).primitive_representative(method="cf")
sage: el
(-T^2*S*T^2*S*T*S*T^(-2)*S*T*S*T*S*T^2*S, T*S)
sage: (el[0]).is_primitive()
True
sage: el = (G.U()^4*G.S()*G.V(2)).acton(-G.V(2)^3*G.V(6)^2*G.V(3)).primitive_
↪representative(method="cf")
sage: el
(-T^2*S*T^2*S*T^2*S*T*S*T^(-2)*S*T*S*T*S, T*S*T*S*T*S*T^2*S)
sage: (el[0]).is_primitive()
True
sage: el = (G.V(1)^5*G.V(2)*G.V(3)^3).primitive_representative(method="cf")
sage: el
(T^2*S*T*S*T^2*S*T*S*T^2*S*T*S*T^7*S, T^6*S)
sage: (el[0]).is_primitive()
True
sage: el = (G.V(2)*G.V(3)).acton(G.U()^6).primitive_representative(method="cf
↪")
sage: el
(T*S, -T*S*T^2*S*T*S*T)
sage: (el[0]).is_primitive()
True

sage: G.element_repr_method("block")
sage: el = G.T().primitive_representative()
sage: (el[0]).is_primitive()
True
sage: el = G.V(2).acton(G.T(-3)).primitive_representative()
sage: el
((-S*T^(-1)*S) * (V(6)) * (-S*T^(-1)*S)^(-1), (T^(-1)) * (V(1)) * (T^(-1))^(-
↪1))
sage: (el[0]).is_primitive()
True
sage: el = (-G.V(2)).primitive_representative()
sage: el
((T*S*T) * (V(2)) * (T*S*T)^(-1), (T*S*T) * (V(2)) * (T*S*T)^(-1))
sage: (el[0]).is_primitive()
True
sage: el = (-G.V(2)^3*G.V(6)^2*G.V(3)).primitive_representative()
sage: el
(V(2)^3*V(6)^2*V(3), 1)
sage: (el[0]).is_primitive()
True
sage: el = (G.U()^4*G.S()*G.V(2)).acton(-G.V(2)^3*G.V(6)^2*G.V(3)).primitive_
↪representative()
sage: el
(V(2)^3*V(6)^2*V(3), (T*S*T*S*T*S*T) * (V(2)*V(4)) * (T*S*T*S*T*S*T)^(-1))
sage: (el[0]).is_primitive()
True
sage: el = (G.V(1)^5*G.V(2)*G.V(3)^3).primitive_representative()
sage: el
(V(3)^3*V(1)^5*V(2), (T^6*S*T) * (V(1)^5*V(2)) * (T^6*S*T)^(-1))
sage: (el[0]).is_primitive()
True

sage: G.element_repr_method("default")
sage: el = G.I().primitive_representative()
```

```
sage: el
(
[1 0]  [1 0]
[0 1], [0 1]
)
sage: (el[0]).is_primitive()
True

sage: el = G.U().primitive_representative()
sage: el
(
[lam  -1]  [1 0]
[  1   0], [0 1]
)
sage: (el[0]).is_primitive()
True
sage: el = (-G.S()).primitive_representative()
sage: el
(
[ 0 -1]  [-1  0]
[ 1  0], [ 0 -1]
)
sage: (el[0]).is_primitive()
True
sage: el = (G.V(2)*G.V(3)).acton(G.U()^6).primitive_representative()
sage: el
(
[lam  -1]  [-2*lam^2 - 2*lam + 2 -2*lam^2 - 2*lam + 1]
[  1   0], [         -2*lam^2 + 1   -2*lam^2 - lam + 2]
)
sage: (el[0]).is_primitive()
True
```

rational_period_function (*k*)

The method assumes that `self` is hyperbolic.

Return the rational period function of weight `k` for the primitive conjugacy class of `self`.

A $rational period function$ of weight `k` is a rational function `q` which satisfies: `q + q|S == 0` and `q + q|U + q|U^2 + ...  + q|U^(n-1) == 0`, where `S = self.parent().S()`, `U = self.parent().U()` and `|` is the usual $slash - operator$ of weight k. Note that if `k < 0` then `q` is a polynomial.

This method returns a very basic rational period function associated with the primitive conjugacy class of `self`. The (strong) expectation is that all rational period functions are formed by linear combinations of such functions.

There is also a close relation with modular integrals of weight `2-k` and sometimes `2-k` is used for the weight instead of `k`.

Warning: The case `n=infinity` is not verified at all and probably wrong!

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.hecke_triangle_groups import␣
↪HeckeTriangleGroup
sage: G = HeckeTriangleGroup(n=5)
sage: S = G.S()
sage: U = G.U()
```

```
sage: def is_rpf(f, k=None):
....:     if not f + S.slash(f, k=k) == 0:
....:         return False
....:     if not sum([(U^m).slash(f, k=k) for m in range(G.n())]) == 0:
....:         return False
....:     return True

sage: z = PolynomialRing(G.base_ring(), 'z').gen()
sage: uniq([ is_rpf(1 - z^(-k), k=k) for k in range(-6, 6, 2)])    # long time
[True]
sage: [is_rpf(1/z, k=k) for k in range(-6, 6, 2)]
[False, False, False, False, True, False]

sage: el = G.V(2)
sage: el.is_hecke_symmetric()
False
sage: rpf = el.rational_period_function(-4)
sage: is_rpf(rpf) == is_rpf(rpf, k=-4)
True
sage: is_rpf(rpf)
True
sage: is_rpf(rpf, k=-6)
False
sage: is_rpf(rpf, k=2)
False
sage: rpf
-lam*z^4 + lam
sage: rpf = el.rational_period_function(-2)
sage: is_rpf(rpf)
True
sage: rpf
(lam + 1)*z^2 - lam - 1
sage: el.rational_period_function(0) == 0
True
sage: rpf = el.rational_period_function(2)
sage: is_rpf(rpf)
True
sage: rpf
((lam + 1)*z^2 - lam - 1)/(lam*z^4 + (-lam - 2)*z^2 + lam)

sage: el = G.V(3)*G.V(2)^(-1)*G.V(1)*G.V(6)
sage: el.is_hecke_symmetric()
False
sage: rpf = el.rational_period_function(-6)
sage: is_rpf(rpf)
True
sage: rpf
(68*lam + 44)*z^6 + (-24*lam - 12)*z^4 + (24*lam + 12)*z^2 - 68*lam - 44
sage: rpf = el.rational_period_function(-2)
sage: is_rpf(rpf)
True
sage: rpf
(4*lam + 4)*z^2 - 4*lam - 4
sage: el.rational_period_function(0) == 0
True
sage: rpf = el.rational_period_function(2)
sage: is_rpf(rpf) == is_rpf(rpf, k=2)
```

```
True
sage: is_rpf(rpf)
True
sage: rpf.denominator()
(8*lam + 5)*z^8 + (-94*lam - 58)*z^6 + (199*lam + 124)*z^4 + (-94*lam - 58)*z^
↪2 + 8*lam + 5

sage: el = G.V(2)*G.V(3)
sage: el.is_hecke_symmetric()
True
sage: el.rational_period_function(-4) == 0
True
sage: rpf = el.rational_period_function(-2)
sage: is_rpf(rpf)
True
sage: rpf
(8*lam + 4)*z^2 - 8*lam - 4
sage: el.rational_period_function(0) == 0
True
sage: rpf = el.rational_period_function(2)
sage: is_rpf(rpf)
True
sage: rpf.denominator()
(144*lam + 89)*z^8 + (-618*lam - 382)*z^6 + (951*lam + 588)*z^4 + (-618*lam -␣
↪382)*z^2 + 144*lam + 89
sage: el.rational_period_function(4) == 0
True
```

reduce (*primitive=True*)

Return a reduced version of `self` (with the same the same fixed points). Also see *is_reduced()* .

If `self` is elliptic (or +- the identity) the result is never reduced (by definition). Instead a more canonical conjugation representative of `self` (resp. it's primitive part) is choosen.

Warning: The case `n=infinity` is not verified at all and probably wrong!

INPUT:

- **`primitive` – If `True` (default) then a primitive** representative for `self` is returned.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.hecke_triangle_groups import␣
↪HeckeTriangleGroup
sage: G = HeckeTriangleGroup(n=7)
sage: print(G.T().reduce().string_repr("basic"))
S*T^(-1)*S*T^(-1)*S*T*S
sage: G.T().reduce().is_reduced(require_hyperbolic=False)
True
sage: print(G.V(2).acton(-G.T(-3)).reduce().string_repr("basic"))
-T*S*T^(-1)*S*T^(-1)
sage: print(G.V(2).acton(-G.T(-3)).reduce(primitive=False).string_repr("basic
↪"))
T*S*T^(-3)*S*T^(-1)
sage: print((-G.V(2)).reduce().string_repr("basic"))
T^2*S
sage: (-G.V(2)).reduce().is_reduced()
True
sage: print((-G.V(2)^3*G.V(6)^2*G.V(3)).reduce().string_repr("block"))
(-S*T^(-1)) * (V(2)^3*V(6)^2*V(3)) * (-S*T^(-1))^(-1)
```

```
sage: (-G.V(2)^3*G.V(6)^2*G.V(3)).reduce().is_reduced()
True

sage: print((-G.I()).reduce().string_repr("block"))
1
sage: print(G.U().reduce().string_repr("block"))
U
sage: print((-G.S()).reduce().string_repr("block"))
S
sage: print((G.V(2)*G.V(3)).acton(G.U()^6).reduce().string_repr("block"))
U
sage: print((G.V(2)*G.V(3)).acton(G.U()^6).reduce(primitive=False).string_
↪repr("block"))
-U^(-1)
```

reduced_elements ( )

Return the cycle of reduced elements in the (primitive) conjugacy class of `self` .

I.e. the set (cycle) of all reduced elements which are conjugate to `self.primitive_part()` . E.g. `self.primitive_representative().reduce()` .

Also see *is_reduced()* . In particular the result of this method only depends on the (primitive) conjugacy class of `self` .

The method assumes that `self` is hyperbolic or parabolic.

Warning: The case `n=infinity` is not verified at all and probably wrong!

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.hecke_triangle_groups import␣
↪HeckeTriangleGroup
sage: G = HeckeTriangleGroup(n=5)
sage: G.element_repr_method("basic")

sage: el = G.V(1)
sage: el.continued_fraction()
((0, 1), (1, 1, 2))
sage: R = el.reduced_elements()
sage: R
[T*S*T*S*T^2*S, T*S*T^2*S*T*S, -T*S*T^(-1)*S*T^(-1)]
sage: [v.continued_fraction() for v in R]
[((), (1, 1, 2)), ((), (1, 2, 1)), ((), (2, 1, 1))]

sage: el = G.V(3)*G.V(2)^(-1)*G.V(1)*G.V(6)
sage: el.continued_fraction()
((1,), (3,))
sage: R = el.reduced_elements()
sage: [v.continued_fraction() for v in R]
[((), (3,))]

sage: G.element_repr_method("default")
```

root_extension_embedding (*K=None*)

Return the correct embedding from the root extension field to `K` .

INPUT:

- **`K` – A field to which we want the (correct) embeddding.** If `K=None` (default) then `AlgebraicField()` is used for elliptic elements and `AlgebraicRealField()`

otherwise.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.hecke_triangle_groups import␣
↪HeckeTriangleGroup
sage: G = HeckeTriangleGroup(n=infinity)

sage: fp = (-G.S()).fixed_points()[0]
sage: alg_fp = (-G.S()).root_extension_embedding()(fp)
sage: alg_fp
1*I
sage: alg_fp == (-G.S()).fixed_points(embedded=True)[0]
True

sage: fp = (-G.V(2)).fixed_points()[1]
sage: alg_fp = (-G.V(2)).root_extension_embedding()(fp)
sage: alg_fp
-1.732050807568...?
sage: alg_fp == (-G.V(2)).fixed_points(embedded=True)[1]
True

sage: fp = (-G.V(2)).fixed_points()[0]
sage: alg_fp = (-G.V(2)).root_extension_embedding()(fp)
sage: alg_fp
1.732050807568...?
sage: alg_fp == (-G.V(2)).fixed_points(embedded=True)[0]
True

sage: G = HeckeTriangleGroup(n=7)

sage: fp = (-G.S()).fixed_points()[1]
sage: alg_fp = (-G.S()).root_extension_embedding()(fp)
sage: alg_fp
0.?... - 1.000000000000...?*I
sage: alg_fp == (-G.S()).fixed_points(embedded=True)[1]
True

sage: fp = (-G.U()^4).fixed_points()[0]
sage: alg_fp = (-G.U()^4).root_extension_embedding()(fp)
sage: alg_fp
0.9009688679024...? + 0.4338837391175...?*I
sage: alg_fp == (-G.U()^4).fixed_points(embedded=True)[0]
True

sage: (-G.U()^4).root_extension_embedding(CC)(fp)
0.900968867902... + 0.433883739117...*I
sage: (-G.U()^4).root_extension_embedding(CC)(fp).parent()
Complex Field with 53 bits of precision

sage: fp = (-G.V(5)).fixed_points()[1]
sage: alg_fp = (-G.V(5)).root_extension_embedding()(fp)
sage: alg_fp
-0.6671145837954...?
sage: alg_fp == (-G.V(5)).fixed_points(embedded=True)[1]
True
```

root_extension_field ( )

Return a field extension which contains the fixed points of `self`. Namely the root extension field of the parent for the discriminant of `self`. Also see the parent method `root_extension_field(D)` and *`root_extension_embedding()`* (which provides the correct embedding).

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.hecke_triangle_groups import␣
↪HeckeTriangleGroup
sage: G = HeckeTriangleGroup(n=infinity)
sage: G.V(3).discriminant()
32
sage: G.V(3).root_extension_field() == G.root_extension_field(32)
True
sage: G.T().root_extension_field() == G.root_extension_field(G.T().
↪discriminant()) == G.base_field()
True
sage: (G.S()).root_extension_field() == G.root_extension_field(G.S().
↪discriminant())
True

sage: G = HeckeTriangleGroup(n=7)
sage: D = G.V(3).discriminant()
sage: D
4*lam^2 + 4*lam - 4
sage: G.V(3).root_extension_field() == G.root_extension_field(D)
True
sage: G.U().root_extension_field() == G.root_extension_field(G.U().
↪discriminant())
True
sage: G.V(1).root_extension_field() == G.base_field()
True
```

sign ( )

Return the sign element/matrix (+- identity) of `self`. The sign is given by the sign of the trace. if the trace is zero it is instead given by the sign of the lower left entry.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.hecke_triangle_groups import␣
↪HeckeTriangleGroup
sage: G = HeckeTriangleGroup(n=7)
sage: (-G.T(-1)).sign()
[-1  0]
[ 0 -1]
sage: G.S().sign()
[1 0]
[0 1]
sage: (-G.S()).sign()
[-1  0]
[ 0 -1]
sage: (G.U()^6).sign()
[-1  0]
[ 0 -1]

sage: G = HeckeTriangleGroup(n=8)
sage: (G.U()^4).trace()
0
sage: (G.U()^4).sign()
[1 0]
[0 1]
```

```
sage: (G.U()^(-4)).sign()
[-1  0]
[ 0 -1]
```

simple_elements ( )

Return all simple elements in the primitive conjugacy class of `self`.

I.e. the set of all simple elements which are conjugate to `self.primitive_part()`.

Also see *`is_simple()`*. In particular the result of this method only depends on the (primitive) conjugacy class of `self`.

The method assumes that `self` is hyperbolic.

Warning: The case `n=infinity` is not verified at all and probably wrong!

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.hecke_triangle_groups import␣
↪HeckeTriangleGroup
sage: G = HeckeTriangleGroup(n=5)

sage: el = G.V(2)
sage: el.continued_fraction()
((1,), (2,))
sage: R = el.simple_elements()
sage: R
[
[lam lam]
[  1 lam]
]
sage: R[0].is_simple()
True

sage: el = G.V(3)*G.V(2)^(-1)*G.V(1)*G.V(6)
sage: el.continued_fraction()
((1,), (3,))
sage: R = el.simple_elements()
sage: R
[
[    2*lam 2*lam + 1]  [      lam 2*lam + 1]
[        1       lam], [        1     2*lam]
]
sage: [v.is_simple() for v in R]
[True, True]

sage: el = G.V(1)^2*G.V(2)*G.V(4)
sage: el.discriminant()
135*lam + 86
sage: R = el.simple_elements()
sage: R
[
[    3*lam 3*lam + 2]  [8*lam + 3 3*lam + 2]  [5*lam + 2 9*lam + 6]
[3*lam + 4 6*lam + 3], [  lam + 2       lam], [  lam + 2 4*lam + 1],
[2*lam + 1 7*lam + 4]
[  lam + 2 7*lam + 2]
]

This agrees with the results (p.16) from Culp-Ressler on
binary quadratic forms for hecke triangle groups:
```

```
sage: [v.continued_fraction() for v in R]
[((1,), (1, 1, 4, 2)),
((3,), (2, 1, 1, 4)),
((2,), (2, 1, 1, 4)),
((1,), (2, 1, 1, 4))]
```

simple_fixed_point_set (*extended=True*)

Return a set of all attracting fixed points in the conjugacy class of the primitive part of `self`.

If `extended=True` (default) then also `S.acton(alpha)` are added for `alpha` in the set.

This is a so called $irreduciblesystemofpoles$ for rational period functions for the parent group. I.e. the fixed points occur as a irreducible part of the non-zero pole set of some rational period function and all pole sets are given as a union of such irreducible systems of poles.

The method assumes that `self` is hyperbolic.

Warning: The case `n=infinity` is not verified at all and probably wrong!

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.hecke_triangle_groups import␣
↪HeckeTriangleGroup
sage: G = HeckeTriangleGroup(n=5)

sage: el = G.V(2)
sage: el.simple_fixed_point_set()
{1/2*e, (-1/2*lam + 1/2)*e}
sage: el.simple_fixed_point_set(extended=False)
{1/2*e}

sage: el = G.V(3)*G.V(2)^(-1)*G.V(1)*G.V(6)
sage: el.simple_fixed_point_set()
{(-lam + 3/2)*e - 1/2*lam + 1, 1/2*e + 1/2*lam, (-lam + 3/2)*e + 1/2*lam - 1,␣
↪1/2*e - 1/2*lam}
sage: el.simple_fixed_point_set(extended=False)
{1/2*e + 1/2*lam, 1/2*e - 1/2*lam}
```

slash (*f*, *tau=None*, *k=None*)

Return the $slash-operator$ of weight `k` to applied to `f`, evaluated at `tau`. I.e. `(f|_k[self])(tau)` .

INPUT:

- **`f` – A function in `tau` (or an object for which** evaluation at `self.acton(tau)` makes sense.
- **`tau` – Where to evaluate the result.** This should be a valid argument for *`acton()`*.

  If `tau` is a point of `HyperbolicPlane()` then its coordinates in the upper half plane model are used.

  Default: `None` in which case `f` has to be a rational function / polynomial in one variable and the generator of the polynomial ring is used for `tau`. That way `slash` acts on rational functions / polynomials.
- `k` – An even integer.

  Default: `None` in which case `f` either has to be a rational function / polynomial in one variable (then -degree is used). Or `f` needs to have a `weight` attribute which is then used.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.hecke_triangle_groups import␣
↪HeckeTriangleGroup
sage: from sage.modular.modform_hecketriangle.space import ModularForms
sage: G = HeckeTriangleGroup(n=5)
sage: E4 = ModularForms(group=G, k=4, ep=1).E4()
sage: z = CC(-1/(-1/(2*i+30)-1))
sage: (G.S()).slash(E4, z)
32288.0558881... - 118329.856601...*I
sage: (G.V(2)*G.V(3)).slash(E4, z)
32288.0558892... - 118329.856603...*I
sage: E4(z)
32288.0558881... - 118329.856601...*I

sage: z = HyperbolicPlane().PD().get_point(CC(-I/2 + 1/8))
sage: (G.V(2)*G.V(3)).slash(E4, z)
-(21624.437... - 12725.035...*I)/((0.610... + 0.324...*I)*sqrt(5) + 2.720...␣
↪+ 0.648...*I)^4

sage: z = PolynomialRing(G.base_ring(), 'z').gen()
sage: rat = z^2 + 1/(z-G.lam())
sage: dr = rat.numerator().degree() - rat.denominator().degree()
sage: G.S().slash(rat) == G.S().slash(rat, tau=None, k=-dr)
True
sage: G.S().slash(rat)
(z^6 - lam*z^4 - z^3)/(-lam*z^4 - z^3)
sage: G.S().slash(rat, k=0)
(z^4 - lam*z^2 - z)/(-lam*z^4 - z^3)
sage: G.S().slash(rat, k=-4)
(z^8 - lam*z^6 - z^5)/(-lam*z^4 - z^3)
```

string_repr (*method='default'*)

Return a string representation of `self` using the specified `method`. This method is used to represent `self`. The default representation method can be set for the parent with `self.parent().element_repr_method(method)`.

INPUT:

- `method` – `default` : Use the usual representation method for matrix group elements.

  `basic` : The representation is given as a word in `S` and powers of `T`. Note: If `S`, `T` are defined accordingly the output can be used/evaluated directly to recover `self`.

  `conj` : The conjugacy representative of the element is represented as a word in powers of the basic blocks, together with an unspecified conjugation matrix.

  `block` : Same as `conj` but the conjugation matrix is specified as well. Note: Assuming `S`, `T`, `U`, `V` are defined accordingly the output can directly be used/evaluated to recover `self`.

Warning: For `n=infinity` the methods `conj` and `block` are not verified at all and are probably wrong!

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.hecke_triangle_groups import␣
↪HeckeTriangleGroup
sage: G = HeckeTriangleGroup(n=5)
sage: el1 = -G.I()
sage: el2 = G.S()*G.T(3)*G.S()*G.T(-2)
sage: el3 = G.V(2)*G.V(3)^2*G.V(4)^3
```

```
sage: el4 = G.U()^4
sage: el5 = (G.V(2)*G.T()).acton(-G.S())

sage: el4.string_repr(method="basic")
'S*T^(-1)'

sage: G.element_repr_method("default")
sage: el1
[-1  0]
[ 0 -1]
sage: el2
[         -1       2*lam]
[      3*lam -6*lam - 7]
sage: el3
[34*lam + 19    5*lam + 4]
[27*lam + 18    5*lam + 2]
sage: el4
[    0    -1]
[    1 -lam]
sage: el5
[-7*lam - 4  9*lam + 6]
[-4*lam - 5  7*lam + 4]

sage: G.element_repr_method("basic")
sage: el1
-1
sage: el2
S*T^3*S*T^(-2)
sage: el3
-T*S*T*S*T^(-1)*S*T^(-2)*S*T^(-4)*S
sage: el4
S*T^(-1)
sage: el5
T*S*T^2*S*T^(-2)*S*T^(-1)

sage: G.element_repr_method("conj")
sage: el1
[-1]
sage: el2
[-V(4)^2*V(1)^3]
sage: el3
[V(3)^2*V(4)^3*V(2)]
sage: el4
[-U^(-1)]
sage: el5
[-S]

sage: G.element_repr_method("block")
sage: el1
-1
sage: el2
-(S*T^3) * (V(4)^2*V(1)^3) * (S*T^3)^(-1)
sage: el3
(T*S*T) * (V(3)^2*V(4)^3*V(2)) * (T*S*T)^(-1)
sage: el4
-U^(-1)
sage: el5
-(T*S*T^2) * (S) * (T*S*T^2)^(-1)
```

```
sage: G.element_repr_method("default")

sage: G = HeckeTriangleGroup(n=infinity)
sage: el = G.S()*G.T(3)*G.S()*G.T(-2)
sage: print(el.string_repr())
[ -1   4]
[  6 -25]
sage: print(el.string_repr(method="basic"))
S*T^3*S*T^(-2)
```

trace ( )

Return the trace of `self`, which is the sum of the diagonal entries.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.hecke_triangle_groups import␣
↪HeckeTriangleGroup
sage: G = HeckeTriangleGroup(n=7)
sage: G.U().trace()
lam
sage: G.S().trace()
0
```

word_S_T ( )

Decompose `self` into a product of the generators `S` and `T` of its parent, together with a sign correction matrix, namely: `self = sgn * prod(L)`.

Warning: If `self` is +- the identity `prod(L)` is an empty product which produces `1` instead of the identity matrix.

OUTPUT:

The function returns a tuple `(L,sgn)` where the entries of `L` are either the generator `S` or a non-trivial integer power of the generator `T`. `sgn` is +- the identity.

If this decomposition is not possible a `TypeError` is raised.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.hecke_triangle_groups import␣
↪HeckeTriangleGroup
sage: G = HeckeTriangleGroup(n=17)
sage: (-G.I()).word_S_T()[0]
()
sage: (-G.I()).word_S_T()[1]
[-1  0]
[ 0 -1]
sage: (L, sgn) = (-G.V(2)).word_S_T()
sage: L
(
[  1 lam]  [ 0 -1]  [  1 lam]
[  0   1], [ 1  0], [  0   1]
)
sage: sgn == -G.I()
True
sage: -G.V(2) == sgn * prod(L)
True
sage: (L, sgn) = G.U().word_S_T()
sage: L
```

```
(
[  1 lam]  [ 0 -1]
[  0   1], [ 1  0]
)
sage: sgn == G.I()
True
sage: G.U() == sgn * prod(L)
True

sage: G = HeckeTriangleGroup(n=infinity)
sage: (L, sgn) = (-G.V(2)*G.V(3)).word_S_T()
sage: L
(
[1 2]  [ 0 -1]  [1 4]  [ 0 -1]  [1 2]  [ 0 -1]  [1 2]
[0 1], [ 1  0], [0 1], [ 1  0], [0 1], [ 1  0], [0 1]
)
sage: -G.V(2)*G.V(3) == sgn * prod(L)
True
```

sage.modular.modform_hecketriangle.hecke_triangle_group_element. **coerce_AA** (*p*)

Return the argument first coerced into `AA` and then simplified. This leads to a major performance gain with some operations.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.hecke_triangle_group_element import␣
↪coerce_AA
sage: p = (791264*AA(2*cos(pi/8))^2 - 463492).sqrt()
sage: AA(p)._exact_field()
Number Field in a with defining polynomial y^8 ... with a in ...
sage: coerce_AA(p)._exact_field()
Number Field in a with defining polynomial y^4 - 1910*y^2 - 3924*y + 681058 with␣
↪a in 39.710518724...?
```

sage.modular.modform_hecketriangle.hecke_triangle_group_element. **cyclic_representative** (*L*)

Return a unique representative among all cyclic permutations of the given list/tuple.

INPUT:

- `L` – A list or tuple.

OUTPUT:

The maximal element among all cyclic permutations with respect to lexicographical ordering.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.hecke_triangle_group_element import␣
↪cyclic_representative
sage: cyclic_representative((1,))
(1,)
sage: cyclic_representative((2,2))
(2, 2)
sage: cyclic_representative((1,2,1,2))
(2, 1, 2, 1)
sage: cyclic_representative((1,2,3,2,3,1))
(3, 2, 3, 1, 1, 2)
```

CHAPTER

TEN

ANALYTIC TYPES OF MODULAR FORMS.

Propoerties of modular forms and their generalizations are assembled into one partially ordered set. See :class:`AnalyticType` for a list of handled properties.

AUTHORS:

- Jonas Jermann (2013): initial version

class `sage.modular.modform_hecketriangle.analytic_type.`**`AnalyticType`**

Bases: `sage.combinat.posets.lattices.FiniteLatticePoset`

Container for all possible analytic types of forms and/or spaces.

The `analytic type` of forms spaces or rings describes all possible occuring basic `analytic properties` of elements in the space/ring (or more).

For ambient spaces/rings this means that all elements with those properties (and the restrictions of the space/ring) are contained in the space/ring.

The analytic type of an element is the analytic type of its minimal ambient space/ring.

The basic `analytic properties` are:

- **`quasi` - Whether the element is quasi modular (and not modular)** or modular.
- **`mero` - `meromorphic` : If the element is meromorphic** and meromorphic at infinity.
- **`weak` - `weakly holomorphic` : If the element is holomorphic** and meromorphic at infinity.
- **`holo` - `holomorphic` : If the element is holomorphic and** holomorphic at infinity.
- **`cusp` - `cuspidal` : If the element additionally has a positive** order at infinity.

The `zero` elements/property have no analytic properties (or only `quasi`).

For ring elements the property describes whether one of its homogeneous components satisfies that property and the "union" of those properties is returned as the `analytic type` .

Similarly for quasi forms the property describes whether one of its quasi components satisfies that property.

There is a (natural) partial order between the basic properties (and analytic types) given by "inclusion". We name the analytic type according to its maximal analytic properties.

For $n = 3$ the quasi form `el = E6 -E2^3` has the quasi components `E6` which is `holomorphic` and `E2^3` which is `quasi holomorphic` . So the analytic type of `el` is `quasi holomorphic` despite the fact that the sum (`el`) describes a function which is zero at infinity.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.space import QuasiModularForms
sage: x,y,z,d = var("x,y,z,d")
sage: el = QuasiModularForms(n=3, k=6, ep=-1)(y-z^3)
sage: el.analytic_type()
quasi modular

Similarly the type of the ring element ``el2 = E4/Delta - E6/Delta`` is
``weakly holomorphic`` despite the fact that the sum (``el2``) describes
a function which is holomorphic at infinity.

sage: from sage.modular.modform_hecketriangle.graded_ring import␣
↪WeakModularFormsRing
sage: x,y,z,d = var("x,y,z,d")
sage: el2 = WeakModularFormsRing(n=3)(x/(x^3-y^2)-y/(x^3-y^2))
sage: el2.analytic_type()
weakly holomorphic modular
```

Element

alias of *AnalyticTypeElement*

base_poset ( )

Return the base poset from which everything of `self` was constructed. Elements of the base poset correspond to the basic `analytic properties`.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.analytic_type import␣
↪AnalyticType
sage: from sage.combinat.posets.posets import FinitePoset
sage: AT = AnalyticType()
sage: P = AT.base_poset()
sage: P
Finite poset containing 5 elements with distinguished linear extension
sage: isinstance(P, FinitePoset)
True

sage: P.is_lattice()
False
sage: P.is_finite()
True
sage: P.cardinality()
5
sage: P.is_bounded()
False
sage: P.list()
[cusp, holo, weak, mero, quasi]

sage: len(P.relations())
11
sage: P.cover_relations()
[[cusp, holo], [holo, weak], [weak, mero]]
sage: P.has_top()
False
sage: P.has_bottom()
False
```

lattice_poset ( )

Return the underlying lattice poset of `self`.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.analytic_type import␣
↪AnalyticType
sage: AnalyticType().lattice_poset()
Finite lattice containing 10 elements
```

class `sage.modular.modform_hecketriangle.analytic_type.`**`AnalyticTypeElement`** (*poset*, *element*, *vertex*)

Bases: `sage.combinat.posets.elements.LatticePosetElement`

Analytic types of forms and/or spaces.

An analytic type element describes what basic analytic properties are contained/included in it.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.analytic_type import (AnalyticType,␣
↪AnalyticTypeElement)
sage: from sage.combinat.posets.elements import LatticePosetElement
sage: AT = AnalyticType()
sage: el = AT(["quasi", "cusp"])
sage: el
quasi cuspidal
sage: isinstance(el, AnalyticTypeElement)
True
sage: isinstance(el, LatticePosetElement)
True
sage: el.parent() == AT
True
sage: el.element
{cusp, quasi}
sage: from sage.sets.set import Set_object_enumerated
sage: isinstance(el.element, Set_object_enumerated)
True
sage: el.element[0]
cusp
sage: el.element[0].parent() == AT.base_poset()
True

sage: el2 = AT("holo")
sage: sum = el + el2
sage: sum
quasi modular
sage: sum.element
{holo, cusp, quasi}
sage: el * el2
cuspidal
```

`analytic_name` ( )

Return a string representation of the analytic type.

sage: from sage.modular.modform_hecketriangle.analytic_type import AnalyticType sage: AT = AnalyticType() sage: AT(["quasi", "weak"]).analytic_name() 'quasi weakly holomorphic modular' sage: AT(["quasi", "cusp"]).analytic_name() 'quasi cuspidal' sage: AT(["quasi"]).analytic_name() 'zero' sage: AT([]).analytic_name() 'zero'

analytic_space_name ( )

Return the (analytic part of the) name of a space with the analytic type of `self`.

This is used for the string representation of such spaces.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.analytic_type import␣
↪AnalyticType
sage: AT = AnalyticType()
sage: AT(["quasi", "weak"]).analytic_space_name()
'QuasiWeakModular'
sage: AT(["quasi", "cusp"]).analytic_space_name()
'QuasiCusp'
sage: AT(["quasi"]).analytic_space_name()
'Zero'
sage: AT([]).analytic_space_name()
'Zero'
```

extend_by (*extend_type*)

Return a new analytic type which contains all analytic properties specified either in `self` or in `extend_type`.

INPUT:

- **`extend_type` – An analytic type or something which is** convertable to an analytic type.

OUTPUT:

The new extended analytic type.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.analytic_type import␣
↪AnalyticType
sage: AT = AnalyticType()
sage: el = AT(["quasi", "cusp"])
sage: el2 = AT("holo")

sage: el.extend_by(el2)
quasi modular
sage: el.extend_by(el2) == el + el2
True
```

latex_space_name ( )

Return the short (analytic part of the) name of a space with the analytic type of `self` for usage with latex.

This is used for the latex representation of such spaces.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.analytic_type import␣
↪AnalyticType
sage: AT = AnalyticType()
sage: AT("mero").latex_space_name()
'\\tilde{M}'
sage: AT("weak").latex_space_name()
'M^!'
sage: AT(["quasi", "cusp"]).latex_space_name()
'QC'
sage: AT([]).latex_space_name()
'Z'
```

reduce_to (*reduce_type*)

Return a new analytic type which contains only analytic properties specified in both `self` and `reduce_type`.

INPUT:

- **`reduce_type` – An analytic type or something which is** convertable to an analytic type.

OUTPUT:

The new reduced analytic type.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.analytic_type import AnalyticType
sage: AT = AnalyticType()
sage: el = AT(["quasi", "cusp"])
sage: el2 = AT("holo")

sage: el.reduce_to(el2)
cuspidal
sage: el.reduce_to(el2) == el * el2
True
```

CHAPTER

ELEVEN

GRADED RINGS OF MODULAR FORMS FOR HECKE TRIANGLE GROUPS

AUTHORS:

- Jonas Jermann (2013): initial version

class `sage.modular.modform_hecketriangle.graded_ring.`**`CuspFormsRing`**`(`*group*, *base_ring*, *red_hom*, *n*`)`

Bases: *`sage.modular.modform_hecketriangle.abstract_ring.FormsRing_abstract`*, `sage.rings.ring.CommutativeAlgebra`, `sage.structure.unique_representation.UniqueRepresen`

Graded ring of (Hecke) cusp forms for the given group and base ring

class `sage.modular.modform_hecketriangle.graded_ring.`**`MeromorphicModularFormsRing`**`(`*group*, *base_ring*, *red_hom*, *n*`)`

Bases: *`sage.modular.modform_hecketriangle.abstract_ring.FormsRing_abstract`*, `sage.rings.ring.CommutativeAlgebra`, `sage.structure.unique_representation.UniqueRepresen`

Graded ring of (Hecke) meromorphic modular forms for the given group and base ring

class `sage.modular.modform_hecketriangle.graded_ring.`**`ModularFormsRing`**`(`*group*, *base_ring*, *red_hom*, *n*`)`

Bases: *`sage.modular.modform_hecketriangle.abstract_ring.FormsRing_abstract`*, `sage.rings.ring.CommutativeAlgebra`, `sage.structure.unique_representation.UniqueRepresen`

Graded ring of (Hecke) modular forms for the given group and base ring

class `sage.modular.modform_hecketriangle.graded_ring.`**`QuasiCuspFormsRing`**`(`*group*, *base_ring*, *red_hom*, *n*`)`

Bases: *`sage.modular.modform_hecketriangle.abstract_ring.FormsRing_abstract`*, `sage.rings.ring.CommutativeAlgebra`, `sage.structure.unique_representation.UniqueRepresen`

Graded ring of (Hecke) quasi cusp forms for the given group and base ring.

class `sage.modular.modform_hecketriangle.graded_ring.`**`QuasiMeromorphicModularFormsRing`**(*group*, *base_ri* *red_hom* *n*)

Bases: *`sage.modular.modform_hecketriangle.abstract_ring.FormsRing_abstract`*, `sage.rings.ring.CommutativeAlgebra`, `sage.structure.unique_representation.UniqueRepresen`

Graded ring of (Hecke) quasi meromorphic modular forms for the given group and base ring.

class `sage.modular.modform_hecketriangle.graded_ring.`**`QuasiModularFormsRing`**(*group*, *base_ring*, *red_hom*, *n*)

Bases: *`sage.modular.modform_hecketriangle.abstract_ring.FormsRing_abstract`*, `sage.rings.ring.CommutativeAlgebra`, `sage.structure.unique_representation.UniqueRepresen`

Graded ring of (Hecke) quasi modular forms for the given group and base ring

class `sage.modular.modform_hecketriangle.graded_ring.`**`QuasiWeakModularFormsRing`**(*group*, *base_ring*, *red_hom*, *n*)

Bases: *`sage.modular.modform_hecketriangle.abstract_ring.FormsRing_abstract`*, `sage.rings.ring.CommutativeAlgebra`, `sage.structure.unique_representation.UniqueRepresen`

Graded ring of (Hecke) quasi weakly holomorphic modular forms for the given group and base ring.

class `sage.modular.modform_hecketriangle.graded_ring.`**`WeakModularFormsRing`**(*group*, *base_ring*, *red_hom*, *n*)

Bases: *`sage.modular.modform_hecketriangle.abstract_ring.FormsRing_abstract`*, `sage.rings.ring.CommutativeAlgebra`, `sage.structure.unique_representation.UniqueRepresen`

Graded ring of (Hecke) weakly holomorphic modular forms for the given group and base ring

`sage.modular.modform_hecketriangle.graded_ring.`**`canonical_parameters`**(*group*, *base_ring*, *red_hom*, *n=None*)

Return a canonical version of the parameters.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.graded_ring import canonical_
↪parameters
sage: canonical_parameters(4, ZZ, 1)
(Hecke triangle group for n = 4, Integer Ring, True, 4)
sage: canonical_parameters(infinity, RR, 0)
(Hecke triangle group for n = +Infinity, Real Field with 53 bits of precision,␣
↪False, +Infinity)
```

CHAPTER

TWELVE

MODULAR FORMS FOR HECKE TRIANGLE GROUPS

AUTHORS:

- Jonas Jermann (2013): initial version

class sage.modular.modform_hecketriangle.space.**CuspForms** (*group*, *base_ring*, *k*, *ep*, *n*)

Bases: *sage.modular.modform_hecketriangle.abstract_space.FormsSpace_abstract* , sage.modules.module.Module , sage.structure.unique_representation.UniqueRepresentation

Module of (Hecke) cusp forms for the given group, base ring, weight and multiplier

coordinate_vector (*v*)

Return the coordinate vector of v with respect to the basis self.gens() .

INPUT:

- v – An element of self .

OUTPUT:

An element of self.module() , namely the corresponding coordinate vector of v with respect to the basis self.gens() .

The module is the free module over the coefficient ring of self with the dimension of self .

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.space import CuspForms
sage: MF = CuspForms(n=12, k=72/5, ep=-1)
sage: MF.default_prec(4)
sage: MF.dimension()
2
sage: el = MF(MF.f_i()*MF.Delta())
sage: el
q - 1/(288*d)*q^2 - 96605/(1327104*d^2)*q^3 + O(q^4)
sage: vec = el.coordinate_vector()
sage: vec
(1, -1/(288*d))
sage: vec.parent()
Vector space of dimension 2 over Fraction Field of Univariate Polynomial Ring␣
↪in d over Integer Ring
sage: vec.parent() == MF.module()
True
sage: el == vec[0]*MF.gen(0) + vec[1]*MF.gen(1)
True
sage: el == MF.element_from_coordinates(vec)
True
```

```
sage: MF = CuspForms(n=infinity, k=16)
sage: el2 = MF(MF.Delta()*MF.E4())
sage: vec2 = el2.coordinate_vector()
sage: vec2
(1, 5/(8*d), 187/(1024*d^2))
sage: el2 == MF.element_from_coordinates(vec2)
True
```

dimension ()

Return the dimension of `self`.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.space import CuspForms
sage: MF = CuspForms(n=12, k=72/5, ep=1)
sage: MF.dimension()
3
sage: len(MF.gens()) == MF.dimension()
True

sage: CuspForms(n=infinity, k=8).dimension()
1
```

gens ()

Return a basis of `self` as a list of basis elements.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.space import CuspForms
sage: MF=CuspForms(n=12, k=72/5, ep=1)
sage: MF
CuspForms(n=12, k=72/5, ep=1) over Integer Ring
sage: MF.dimension()
3
sage: MF.gens()
[q + 296888795/(10319560704*d^3)*q^4 + O(q^5),
 q^2 + 6629/(221184*d^2)*q^4 + O(q^5),
 q^3 - 25/(96*d)*q^4 + O(q^5)]

sage: MF = CuspForms(n=infinity, k=8, ep=1)
sage: MF.gen(0) == MF.E4()*MF.f_inf()
True
```

class `sage.modular.modform_hecketriangle.space.`**MeromorphicModularForms** (*group*, *base_ring*, *k*, *ep*, *n*)

Bases: *sage.modular.modform_hecketriangle.abstract_space.FormsSpace_abstract*, `sage.modules.module.Module`, `sage.structure.unique_representation.UniqueRepresentation`

Module of (Hecke) meromorphic modular forms for the given group, base ring, weight and multiplier

class `sage.modular.modform_hecketriangle.space.`**ModularForms** (*group*, *base_ring*, *k*, *ep*, *n*)

Bases: *sage.modular.modform_hecketriangle.abstract_space.FormsSpace_abstract*, `sage.modules.module.Module`, `sage.structure.unique_representation.UniqueRepresentation`

Module of (Hecke) modular forms for the given group, base ring, weight and multiplier

coordinate_vector (*v*)

Return the coordinate vector of `v` with respect to the basis `self.gens()` .

INPUT:

- `v` – An element of `self` .

OUTPUT:

An element of `self.module()` , namely the corresponding coordinate vector of `v` with respect to the basis `self.gens()` .

The module is the free module over the coefficient ring of `self` with the dimension of `self` .

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.space import ModularForms
sage: MF = ModularForms(n=6, k=20, ep=1)
sage: MF.dimension()
4
sage: el = MF.E4()^2*MF.Delta()
sage: el
q + 78*q^2 + 2781*q^3 + 59812*q^4 + O(q^5)
sage: vec = el.coordinate_vector()
sage: vec
(0, 1, 13/(18*d), 103/(432*d^2))
sage: vec.parent()
Vector space of dimension 4 over Fraction Field of Univariate Polynomial Ring␣
↪in d over Integer Ring
sage: vec.parent() == MF.module()
True
sage: el == vec[0]*MF.gen(0) + vec[1]*MF.gen(1) + vec[2]*MF.gen(2) +␣
↪vec[3]*MF.gen(3)
True
sage: el == MF.element_from_coordinates(vec)
True

sage: MF = ModularForms(n=infinity, k=8, ep=1)
sage: (MF.E4()^2).coordinate_vector()
(1, 1/(2*d), 15/(128*d^2))
```

dimension ( )

Return the dimension of `self` .

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.space import ModularForms
sage: MF = ModularForms(n=6, k=20, ep=1)
sage: MF.dimension()
4
sage: len(MF.gens()) == MF.dimension()
True

sage: ModularForms(n=infinity, k=8).dimension()
3
```

gens ( )

Return a basis of `self` as a list of basis elements.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.space import ModularForms
sage: MF = ModularForms(n=6, k=20, ep=1)
sage: MF.dimension()
4
sage: MF.gens()
[1 + 360360*q^4 + O(q^5),
 q + 21742*q^4 + O(q^5),
 q^2 + 702*q^4 + O(q^5),
 q^3 - 6*q^4 + O(q^5)]

sage: ModularForms(n=infinity, k=4).gens()
[1 + 240*q^2 + 2160*q^4 + O(q^5), q - 8*q^2 + 28*q^3 - 64*q^4 + O(q^5)]
```

class `sage.modular.modform_hecketriangle.space.`**`QuasiCuspForms`**(*group*, *base_ring*, *k*, *ep*, *n*)

Bases: *sage.modular.modform_hecketriangle.abstract_space.FormsSpace_abstract*, `sage.modules.module.Module`, `sage.structure.unique_representation.UniqueRepresentation`

Module of (Hecke) quasi cusp forms for the given group, base ring, weight and multiplier

`coordinate_vector`(*v*)

Return the coordinate vector of `v` with respect to the basis `self.gens()`.

INPUT:

- `v` – An element of `self`.

OUTPUT:

An element of `self.module()`, namely the corresponding coordinate vector of `v` with respect to the basis `self.gens()`.

The module is the free module over the coefficient ring of `self` with the dimension of `self`.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.space import QuasiCuspForms
sage: MF = QuasiCuspForms(n=6, k=20, ep=1)
sage: MF.dimension()
12
sage: el = MF(MF.E4()^2*MF.Delta() + MF.E4()*MF.E2()^2*MF.Delta())
sage: el
2*q + 120*q^2 + 3402*q^3 + 61520*q^4 + O(q^5)
sage: vec = el.coordinate_vector()     # long time
sage: vec     # long time
(1, 13/(18*d), 103/(432*d^2), 0, 0, 1, 1/(2*d), 0, 0, 0, 0, 0)
sage: vec.parent()     # long time
Vector space of dimension 12 over Fraction Field of Univariate Polynomial␣
↪Ring in d over Integer Ring
sage: vec.parent() == MF.module()     # long time
True
sage: el == MF(sum([vec[l]*MF.gen(l) for l in range(0,12)]))     # long time
True
sage: el == MF.element_from_coordinates(vec)     # long time
True
sage: MF.gen(1).coordinate_vector() == vector([0, 1, 0, 0, 0, 0, 0, 0, 0, 0,␣
↪0, 0])     # long time
True

sage: MF = QuasiCuspForms(n=infinity, k=10, ep=-1)
```

```
sage: el2 = MF(MF.E4()*MF.f_inf()*(MF.f_i() - MF.E2()))
sage: el2.coordinate_vector()
(1, -1)
sage: el2 == MF.element_from_coordinates(el2.coordinate_vector())
True
```

dimension ( )

Return the dimension of `self`.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.space import QuasiCuspForms
sage: MF = QuasiCuspForms(n=8, k=46/3, ep=-1)
sage: MF.default_prec(3)
sage: MF.dimension()
7
sage: len(MF.gens()) == MF.dimension()
True

sage: QuasiCuspForms(n=infinity, k=10, ep=-1).dimension()
2
```

gens ( )

Return a basis of `self` as a list of basis elements.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.space import QuasiCuspForms
sage: MF = QuasiCuspForms(n=8, k=46/3, ep=-1)
sage: MF.default_prec(4)
sage: MF.dimension()
7
sage: MF.gens()
[q - 17535/(262144*d^2)*q^3 + O(q^4),
 q^2 - 47/(128*d)*q^3 + O(q^4),
 q - 9/(128*d)*q^2 + 15633/(262144*d^2)*q^3 + O(q^4),
 q^2 - 7/(128*d)*q^3 + O(q^4),
 q - 23/(64*d)*q^2 - 3103/(262144*d^2)*q^3 + O(q^4),
 q - 3/(64*d)*q^2 - 4863/(262144*d^2)*q^3 + O(q^4),
 q - 27/(64*d)*q^2 + 17217/(262144*d^2)*q^3 + O(q^4)]

sage: MF = QuasiCuspForms(n=infinity, k=10, ep=-1)
sage: MF.gens()
[q - 16*q^2 - 156*q^3 - 256*q^4 + O(q^5), q - 60*q^3 - 256*q^4 + O(q^5)]
```

class `sage.modular.modform_hecketriangle.space.`**`QuasiMeromorphicModularForms`** (*group*, *base_ring*, *k*, *ep*, *n*)

Bases: *`sage.modular.modform_hecketriangle.abstract_space.FormsSpace_abstract`*, `sage.modules.module.Module`, `sage.structure.unique_representation.UniqueRepresentation`

Module of (Hecke) quasi meromorphic modular forms for the given group, base ring, weight and multiplier

class `sage.modular.modform_hecketriangle.space.`**`QuasiModularForms`** (*group*, *base_ring*, *k*, *ep*, *n*)

Bases: *sage.modular.modform_hecketriangle.abstract_space.FormsSpace_abstract*, sage.modules.module.Module, sage.structure.unique_representation.UniqueRepresentation

Module of (Hecke) quasi modular forms for the given group, base ring, weight and multiplier

coordinate_vector (*v*)

Return the coordinate vector of v with respect to the basis self.gens().

INPUT:

- v – An element of self.

OUTPUT:

An element of self.module(), namely the corresponding coordinate vector of v with respect to the basis self.gens().

The module is the free module over the coefficient ring of self with the dimension of self.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.space import QuasiModularForms
sage: MF = QuasiModularForms(n=6, k=20, ep=1)
sage: MF.dimension()
22
sage: el = MF(MF.E4()^2*MF.E6()^2 + MF.E4()*MF.E2()^2*MF.Delta() + MF.E2()^
↪3*MF.E4()^2*MF.E6())
sage: el
2 + 25*q - 2478*q^2 - 82731*q^3 - 448484*q^4 + O(q^5)
sage: vec = el.coordinate_vector()     # long time
sage: vec     # long time
(1, 1/(9*d), -11/(81*d^2), -4499/(104976*d^3), 0, 0, 0, 0, 1, 1/(2*d), 1, 5/
↪(18*d), 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)
sage: vec.parent()     # long time
Vector space of dimension 22 over Fraction Field of Univariate Polynomial␣
↪Ring in d over Integer Ring
sage: vec.parent() == MF.module()     # long time
True
sage: el == MF(sum([vec[l]*MF.gen(l) for l in range(0,22)]))     # long time
True
sage: el == MF.element_from_coordinates(vec)     # long time
True
sage: MF.gen(1).coordinate_vector() == vector([0, 1, 0, 0, 0, 0, 0, 0, 0, 0,␣
↪0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])     # long time
True

sage: MF = QuasiModularForms(n=infinity, k=4, ep=1)
sage: el2 = MF.E4() + MF.E2()^2
sage: el2
2 + 160*q^2 + 512*q^3 + 1632*q^4 + O(q^5)
sage: el2.coordinate_vector()
(1, 1/(4*d), 0, 1)
sage: el2 == MF.element_from_coordinates(el2.coordinate_vector())
True
```

dimension ()

Return the dimension of self.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.space import QuasiModularForms
sage: MF = QuasiModularForms(n=5, k=6, ep=-1)
sage: MF.dimension()
3
sage: len(MF.gens()) == MF.dimension()
True
```

gens ( )

Return a basis of `self` as a list of basis elements.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.space import QuasiModularForms
sage: MF = QuasiModularForms(n=5, k=6, ep=-1)
sage: MF.default_prec(2)
sage: MF.gens()
[1 - 37/(200*d)*q + O(q^2),
 1 + 33/(200*d)*q + O(q^2),
 1 - 27/(200*d)*q + O(q^2)]

sage: MF = QuasiModularForms(n=infinity, k=2, ep=-1)
sage: MF.default_prec(2)
sage: MF.gens()
[1 - 24*q + O(q^2), 1 - 8*q + O(q^2)]
```

class `sage.modular.modform_hecketriangle.space.`**QuasiWeakModularForms** (*group*, *base_ring*, *k*, *ep*, *n*)

Bases: *sage.modular.modform_hecketriangle.abstract_space.FormsSpace_abstract*, `sage.modules.module.Module`, `sage.structure.unique_representation.UniqueRepresentation`

Module of (Hecke) quasi weakly holomorphic modular forms for the given group, base ring, weight and multiplier

class `sage.modular.modform_hecketriangle.space.`**WeakModularForms** (*group*, *base_ring*, *k*, *ep*, *n*)

Bases: *sage.modular.modform_hecketriangle.abstract_space.FormsSpace_abstract*, `sage.modules.module.Module`, `sage.structure.unique_representation.UniqueRepresentation`

Module of (Hecke) weakly holomorphic modular forms for the given group, base ring, weight and multiplier

class `sage.modular.modform_hecketriangle.space.`**ZeroForm** (*group*, *base_ring*, *k*, *ep*, *n*)

Bases: *sage.modular.modform_hecketriangle.abstract_space.FormsSpace_abstract*, `sage.modules.module.Module`, `sage.structure.unique_representation.UniqueRepresentation`

Zero Module for the zero form for the given group, base ring weight and multiplier

coordinate_vector (*v*)

Return the coordinate vector of `v` with respect to the basis `self.gens()`.

Since this is the zero module which only contains the zero form the trivial vector in the trivial module of dimension `0` is returned.

INPUT:

- `v` – An element of `self`, i.e. in this case the zero vector.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.space import ZeroForm
sage: MF = ZeroForm(6, CC, 3, -1)
sage: el = MF(0)
sage: el
O(q^5)
sage: vec = el.coordinate_vector()
sage: vec
()
sage: vec.parent()
Vector space of dimension 0 over Fraction Field of Univariate Polynomial Ring␣
↪in d over Complex Field with 53 bits of precision
sage: vec.parent() == MF.module()
True
```

dimension ( )

Return the dimension of `self`. Since this is the zero module `0` is returned.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.space import ZeroForm
sage: ZeroForm(6, CC, 3, -1).dimension()
0
```

gens ( )

Return a basis of `self` as a list of basis elements. Since this is the zero module an empty list is returned.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.space import ZeroForm
sage: ZeroForm(6, CC, 3, -1).gens()
[]
```

`sage.modular.modform_hecketriangle.space.`**canonical_parameters** (*group*, *base_ring*, *k*, *ep*, *n=None*)

Return a canonical version of the parameters.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.space import canonical_parameters
sage: canonical_parameters(5, ZZ, 20/3, int(1))
(Hecke triangle group for n = 5, Integer Ring, 20/3, 1, 5)

sage: canonical_parameters(infinity, ZZ, 2, int(-1))
(Hecke triangle group for n = +Infinity, Integer Ring, 2, -1, +Infinity)
```

CHAPTER

THIRTEEN

SUBSPACES OF MODULAR FORMS FOR HECKE TRIANGLE GROUPS

AUTHORS:

- Jonas Jermann (2013): initial version

sage.modular.modform_hecketriangle.subspace.**ModularFormsSubSpace**(**args*, ***kwargs*)

Create a modular forms subspace generated by the supplied arguments if possible. Instead of a list of generators also multiple input arguments can be used. If `reduce=True` then the corresponding ambient space is choosen as small as possible. If no subspace is available then the ambient space is returned.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.subspace import ModularFormsSubSpace
sage: from sage.modular.modform_hecketriangle.space import ModularForms
sage: MF = ModularForms()
sage: subspace = ModularFormsSubSpace(MF.E4()^3, MF.E6()^2+MF.Delta(), MF.Delta())
sage: subspace
Subspace of dimension 2 of ModularForms(n=3, k=12, ep=1) over Integer Ring
sage: subspace.ambient_space()
ModularForms(n=3, k=12, ep=1) over Integer Ring
sage: subspace.gens()
[1 + 720*q + 179280*q^2 + 16954560*q^3 + 396974160*q^4 + O(q^5), 1 - 1007*q +␣
↪220728*q^2 + 16519356*q^3 + 399516304*q^4 + O(q^5)]
sage: ModularFormsSubSpace(MF.E4()^3-MF.E6()^2, reduce=True).ambient_space()
CuspForms(n=3, k=12, ep=1) over Integer Ring
sage: ModularFormsSubSpace(MF.E4()^3-MF.E6()^2, MF.J_inv()*MF.E4()^3, reduce=True)
WeakModularForms(n=3, k=12, ep=1) over Integer Ring
```

class sage.modular.modform_hecketriangle.subspace.**SubSpaceForms**(*ambient_space*, *basis*, *check*)

Bases: *sage.modular.modform_hecketriangle.abstract_space.FormsSpace_abstract*, sage.modules.module.Module, sage.structure.unique_representation.UniqueRepresentation

Submodule of (Hecke) forms in the given ambient space for the given basis.

basis()

Return the basis of `self` in the ambient space.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.space import ModularForms
sage: MF = ModularForms(n=6, k=20, ep=1)
sage: subspace = MF.subspace([(MF.Delta()*MF.E4()^2).as_ring_element(), MF.
↪gen(0)])
```

```
sage: subspace.basis()
[q + 78*q^2 + 2781*q^3 + 59812*q^4 + O(q^5), 1 + 360360*q^4 + O(q^5)]
sage: subspace.basis()[0].parent() == MF
True
```

change_ambient_space (*new_ambient_space*)

Return a new subspace with the same basis but inside a different ambient space (if possible).

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.space import ModularForms,␣
↪QuasiModularForms
sage: MF = ModularForms(n=6, k=20, ep=1)
sage: subspace = MF.subspace([MF.Delta()*MF.E4()^2, MF.gen(0)])
sage: new_ambient_space = QuasiModularForms(n=6, k=20, ep=1)
sage: subspace.change_ambient_space(new_ambient_space)     # long time
Subspace of dimension 2 of QuasiModularForms(n=6, k=20, ep=1) over Integer␣
↪Ring
```

change_ring (*new_base_ring*)

Return the same space as `self` but over a new base ring `new_base_ring`.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.space import ModularForms
sage: MF = ModularForms(n=6, k=20, ep=1)
sage: subspace = MF.subspace([MF.Delta()*MF.E4()^2, MF.gen(0)])
sage: subspace.change_ring(CC)
Subspace of dimension 2 of ModularForms(n=6, k=20, ep=1) over Complex Field␣
↪with 53 bits of precision
```

contains_coeff_ring ( )

Return whether `self` contains its coefficient ring.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.space import ModularForms
sage: MF = ModularForms(k=0, ep=1, n=8)
sage: subspace = MF.subspace([1])
sage: subspace.contains_coeff_ring()
True
sage: subspace = MF.subspace([])
sage: subspace.contains_coeff_ring()
False
sage: MF = ModularForms(k=0, ep=-1, n=8)
sage: subspace = MF.subspace([])
sage: subspace.contains_coeff_ring()
False
```

coordinate_vector (*v*)

Return the coordinate vector of `v` with respect to the basis `self.gens()`.

INPUT:

- `v` – An element of `self`.

OUTPUT:

The coordinate vector of `v` with respect to the basis `self.gens()`.

Note: The coordinate vector is not an element of `self.module()`.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.space import ModularForms,␣
↪QuasiCuspForms
sage: MF = ModularForms(n=6, k=20, ep=1)
sage: subspace = MF.subspace([(MF.Delta()*MF.E4()^2).as_ring_element(), MF.
↪gen(0)])
sage: subspace.coordinate_vector(MF.gen(0) + MF.Delta()*MF.E4()^2).parent()
Vector space of dimension 2 over Fraction Field of Univariate Polynomial Ring␣
↪in d over Integer Ring
sage: subspace.coordinate_vector(MF.gen(0) + MF.Delta()*MF.E4()^2)
(1, 1)

sage: MF = ModularForms(n=4, k=24, ep=-1)
sage: subspace = MF.subspace([MF.gen(0), MF.gen(2)])
sage: subspace.coordinate_vector(subspace.gen(0)).parent()
Vector space of dimension 2 over Fraction Field of Univariate Polynomial Ring␣
↪in d over Integer Ring
sage: subspace.coordinate_vector(subspace.gen(0))
(1, 0)

sage: MF = QuasiCuspForms(n=infinity, k=12, ep=1)
sage: subspace = MF.subspace([MF.Delta(), MF.E4()*MF.f_inf()*MF.E2()*MF.f_
↪i(), MF.E4()*MF.f_inf()*MF.E2()^2, MF.E4()*MF.f_inf()*(MF.E4()-MF.E2()^2)])
sage: el = MF.E4()*MF.f_inf()*(7*MF.E4() - 3*MF.E2()^2)
sage: subspace.coordinate_vector(el)
(7, 0, -3)
sage: subspace.ambient_coordinate_vector(el)
(7, 21/(8*d), 0, -3)
```

degree ( )

Return the degree of `self`.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.space import ModularForms
sage: MF = ModularForms(n=6, k=20, ep=1)
sage: subspace = MF.subspace([(MF.Delta()*MF.E4()^2).as_ring_element(), MF.
↪gen(0)])
sage: subspace.degree()
4
sage: subspace.degree() == subspace.ambient_space().degree()
True
```

dimension ( )

Return the dimension of `self`.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.space import ModularForms
sage: MF = ModularForms(n=6, k=20, ep=1)
sage: subspace = MF.subspace([(MF.Delta()*MF.E4()^2).as_ring_element(), MF.
↪gen(0)])
sage: subspace.dimension()
2
sage: subspace.dimension() == len(subspace.gens())
True
```

gens ( )

Return the basis of `self`.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.space import ModularForms
sage: MF = ModularForms(n=6, k=20, ep=1)
sage: subspace = MF.subspace([(MF.Delta()*MF.E4()^2).as_ring_element(), MF.
↪gen(0)])
sage: subspace.gens()
[q + 78*q^2 + 2781*q^3 + 59812*q^4 + O(q^5), 1 + 360360*q^4 + O(q^5)]
sage: subspace.gens()[0].parent() == subspace
True
```

rank ()

Return the rank of `self`.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.space import ModularForms
sage: MF = ModularForms(n=6, k=20, ep=1)
sage: subspace = MF.subspace([(MF.Delta()*MF.E4()^2).as_ring_element(), MF.
↪gen(0)])
sage: subspace.rank()
2
sage: subspace.rank() == subspace.dimension()
True
```

`sage.modular.modform_hecketriangle.subspace.`**canonical_parameters** (*ambient_space*, *basis*, *check=True*)

Return a canonical version of the parameters. In particular the list/tuple `basis` is replaced by a tuple of linearly independent elements in the ambient space.

If `check=False` (default: `True`) then `basis` is assumed to already be a basis.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.subspace import canonical_parameters
sage: from sage.modular.modform_hecketriangle.space import ModularForms
sage: MF = ModularForms(n=6, k=12, ep=1)
sage: canonical_parameters(MF, [MF.Delta().as_ring_element(), MF.gen(0), 2*MF.
↪gen(0)])
(ModularForms(n=6, k=12, ep=1) over Integer Ring,
 (q + 30*q^2 + 333*q^3 + 1444*q^4 + O(q^5),
  1 + 26208*q^3 + 530712*q^4 + O(q^5)))
```

CHAPTER

FOURTEEN

SERIES CONSTRUCTOR FOR MODULAR FORMS FOR HECKE TRIANGLE GROUPS

AUTHORS:

- Based on the thesis of John Garrett Leo (2008)
- Jonas Jermann (2013): initial version

class `sage.modular.modform_hecketriangle.series_constructor.`**`MFSeriesConstructor`**`(`*group*, *prec*`)`

Bases: `sage.structure.sage_object.SageObject`, `sage.structure.unique_representation.UniqueR`

Constructor for the Fourier expansion of some (specific, basic) modular forms.

The constructor is used by forms elements in case their Fourier expansion is needed or requested.

`Delta_ZZ``()`

Return the rational Fourier expansion of `Delta`, where the parameter `d` is replaced by `1`.

The Fourier expansion of `Delta` for `d!=1` is given by `d*Delta_ZZ(q/d)`.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.series_constructor import␣
↪MFSeriesConstructor
sage: MFSeriesConstructor(prec=3).Delta_ZZ()
q - 1/72*q^2 + 7/82944*q^3 + O(q^4)
sage: MFSeriesConstructor(group=5, prec=3).Delta_ZZ()
q + 47/200*q^2 + 11367/640000*q^3 + O(q^4)
sage: MFSeriesConstructor(group=5, prec=3).Delta_ZZ().parent()
Power Series Ring in q over Rational Field

sage: MFSeriesConstructor(group=infinity, prec=3).Delta_ZZ()
q + 3/8*q^2 + 63/1024*q^3 + O(q^4)
```

`E2_ZZ``()`

Return the rational Fourier expansion of `E2`, where the parameter `d` is replaced by `1`.

The Fourier expansion of `E2` for `d!=1` is given by `E2_ZZ(q/d)`.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.series_constructor import␣
↪MFSeriesConstructor
sage: MFSeriesConstructor(prec=3).E2_ZZ()
1 - 1/72*q - 1/41472*q^2 + O(q^3)
sage: MFSeriesConstructor(group=5, prec=3).E2_ZZ()
```

```
1 - 9/200*q - 369/320000*q^2 + O(q^3)
sage: MFSeriesConstructor(group=5, prec=3).E2_ZZ().parent()
Power Series Ring in q over Rational Field

sage: MFSeriesConstructor(group=infinity, prec=3).E2_ZZ()
1 - 1/8*q - 1/512*q^2 + O(q^3)
```

E4_ZZ ( )

Return the rational Fourier expansion of `E_4` , where the parameter `d` is replaced by `1` .

The Fourier expansion of `E4` for `d!=1` is given by `E4_ZZ(q/d)` .

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.series_constructor import␣
↪MFSeriesConstructor
sage: MFSeriesConstructor(prec=3).E4_ZZ()
1 + 5/36*q + 5/6912*q^2 + O(q^3)
sage: MFSeriesConstructor(group=5, prec=3).E4_ZZ()
1 + 21/100*q + 483/32000*q^2 + O(q^3)
sage: MFSeriesConstructor(group=5, prec=3).E4_ZZ().parent()
Power Series Ring in q over Rational Field

sage: MFSeriesConstructor(group=infinity, prec=3).E4_ZZ()
1 + 1/4*q + 7/256*q^2 + O(q^3)
```

E6_ZZ ( )

Return the rational Fourier expansion of `E_6` , where the parameter `d` is replaced by `1` .

The Fourier expansion of `E6` for `d!=1` is given by `E6_ZZ(q/d)` .

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.series_constructor import␣
↪MFSeriesConstructor
sage: MFSeriesConstructor(prec=3).E6_ZZ()
1 - 7/24*q - 77/13824*q^2 + O(q^3)
sage: MFSeriesConstructor(group=5, prec=3).E6_ZZ()
1 - 37/200*q - 14663/320000*q^2 + O(q^3)
sage: MFSeriesConstructor(group=5, prec=3).E6_ZZ().parent()
Power Series Ring in q over Rational Field

sage: MFSeriesConstructor(group=infinity, prec=3).E6_ZZ()
1 - 1/8*q - 31/512*q^2 + O(q^3)
```

EisensteinSeries_ZZ (*k*)

Return the rational Fourier expansion of the normalized Eisenstein series of weight `k` , where the parameter `d` is replaced by `1` .

Only arithmetic groups with `n < infinity` are supported!

THe Fourier expansion of the series is given by `EisensteinSeries_ZZ(q/d)` .

INPUT:

- `k` – A non-negative even integer, namely the weight.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.series_constructor import␣
↪MFSeriesConstructor
sage: MFC = MFSeriesConstructor(prec=6)
sage: MFC.EisensteinSeries_ZZ(k=0)
1
sage: MFC.EisensteinSeries_ZZ(k=2)
1 - 1/72*q - 1/41472*q^2 - 1/53747712*q^3 - 7/371504185344*q^4 - 1/
↪106993205379072*q^5 + O(q^6)
sage: MFC.EisensteinSeries_ZZ(k=6)
1 - 7/24*q - 77/13824*q^2 - 427/17915904*q^3 - 7399/123834728448*q^4 - 3647/
↪35664401793024*q^5 + O(q^6)
sage: MFC.EisensteinSeries_ZZ(k=12)
1 + 455/8292*q + 310765/4776192*q^2 + 20150585/6189944832*q^3 + 1909340615/
↪42784898678784*q^4 + 3702799555/12322050819489792*q^5 + O(q^6)
sage: MFC.EisensteinSeries_ZZ(k=12).parent()
Power Series Ring in q over Rational Field

sage: MFC = MFSeriesConstructor(group=4, prec=5)
sage: MFC.EisensteinSeries_ZZ(k=2)
1 - 1/32*q - 5/8192*q^2 - 1/524288*q^3 - 13/536870912*q^4 + O(q^5)
sage: MFC.EisensteinSeries_ZZ(k=4)
1 + 3/16*q + 39/4096*q^2 + 21/262144*q^3 + 327/268435456*q^4 + O(q^5)
sage: MFC.EisensteinSeries_ZZ(k=6)
1 - 7/32*q - 287/8192*q^2 - 427/524288*q^3 - 9247/536870912*q^4 + O(q^5)
sage: MFC.EisensteinSeries_ZZ(k=12)
1 + 63/11056*q + 133119/2830336*q^2 + 2790081/181141504*q^3 + 272631807/
↪185488900096*q^4 + O(q^5)

sage: MFC = MFSeriesConstructor(group=6, prec=5)
sage: MFC.EisensteinSeries_ZZ(k=2)
1 - 1/18*q - 1/648*q^2 - 7/209952*q^3 - 7/22674816*q^4 + O(q^5)
sage: MFC.EisensteinSeries_ZZ(k=4)
1 + 2/9*q + 1/54*q^2 + 37/52488*q^3 + 73/5668704*q^4 + O(q^5)
sage: MFC.EisensteinSeries_ZZ(k=6)
1 - 1/6*q - 11/216*q^2 - 271/69984*q^3 - 1057/7558272*q^4 + O(q^5)
sage: MFC.EisensteinSeries_ZZ(k=12)
1 + 182/151329*q + 62153/2723922*q^2 + 16186807/882550728*q^3 + 381868123/
↪95315478624*q^4 + O(q^5)
```

G_inv_ZZ ()

Return the rational Fourier expansion of `G_inv`, where the parameter `d` is replaced by `1`.

The Fourier expansion of `G_inv` for `d!=1` is given by `d*G_inv_ZZ(q/d)`.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.series_constructor import␣
↪MFSeriesConstructor
sage: MFSeriesConstructor(group=4, prec=3).G_inv_ZZ()
q^-1 - 3/32 - 955/16384*q + O(q^2)
sage: MFSeriesConstructor(group=8, prec=3).G_inv_ZZ()
q^-1 - 15/128 - 15139/262144*q + O(q^2)
sage: MFSeriesConstructor(group=8, prec=3).G_inv_ZZ().parent()
Laurent Series Ring in q over Rational Field

sage: MFSeriesConstructor(group=infinity, prec=3).G_inv_ZZ()
q^-1 - 1/8 - 59/1024*q + O(q^2)
```

J_inv_ZZ ()

Return the rational Fourier expansion of `J_inv`, where the parameter `d` is replaced by `1`.

This is the main function used to determine all Fourier expansions!

The Fourier expansion of `J_inv` for `d!=1` is given by `J_inv_ZZ(q/d)`.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.series_constructor import␣
↪MFSeriesConstructor
sage: MFSeriesConstructor(prec=3).J_inv_ZZ()
q^-1 + 31/72 + 1823/27648*q + O(q^2)
sage: MFSeriesConstructor(group=5, prec=3).J_inv_ZZ()
q^-1 + 79/200 + 42877/640000*q + O(q^2)
sage: MFSeriesConstructor(group=5, prec=3).J_inv_ZZ().parent()
Laurent Series Ring in q over Rational Field

sage: MFSeriesConstructor(group=infinity, prec=3).J_inv_ZZ()
q^-1 + 3/8 + 69/1024*q + O(q^2)
```

f_i_ZZ ( )

Return the rational Fourier expansion of `f_i`, where the parameter `d` is replaced by `1`.

The Fourier expansion of `f_i` for `d!=1` is given by `f_i_ZZ(q/d)`.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.series_constructor import␣
↪MFSeriesConstructor
sage: MFSeriesConstructor(prec=3).f_i_ZZ()
1 - 7/24*q - 77/13824*q^2 + O(q^3)
sage: MFSeriesConstructor(group=5, prec=3).f_i_ZZ()
1 - 13/40*q - 351/64000*q^2 + O(q^3)
sage: MFSeriesConstructor(group=5, prec=3).f_i_ZZ().parent()
Power Series Ring in q over Rational Field

sage: MFSeriesConstructor(group=infinity, prec=3).f_i_ZZ()
1 - 3/8*q + 3/512*q^2 + O(q^3)
```

f_inf_ZZ ( )

Return the rational Fourier expansion of `f_inf`, where the parameter `d` is replaced by `1`.

The Fourier expansion of `f_inf` for `d!=1` is given by `d*f_inf_ZZ(q/d)`.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.series_constructor import␣
↪MFSeriesConstructor
sage: MFSeriesConstructor(prec=3).f_inf_ZZ()
q - 1/72*q^2 + 7/82944*q^3 + O(q^4)
sage: MFSeriesConstructor(group=5, prec=3).f_inf_ZZ()
q - 9/200*q^2 + 279/640000*q^3 + O(q^4)
sage: MFSeriesConstructor(group=5, prec=3).f_inf_ZZ().parent()
Power Series Ring in q over Rational Field

sage: MFSeriesConstructor(group=infinity, prec=3).f_inf_ZZ()
q - 1/8*q^2 + 7/1024*q^3 + O(q^4)
```

f_rho_ZZ ( )

Return the rational Fourier expansion of `f_rho`, where the parameter `d` is replaced by `1`.

The Fourier expansion of `f_rho` for `d!=1` is given by `f_rho_ZZ(q/d)`.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.series_constructor import␣
↪MFSeriesConstructor
sage: MFSeriesConstructor(prec=3).f_rho_ZZ()
1 + 5/36*q + 5/6912*q^2 + O(q^3)
sage: MFSeriesConstructor(group=5, prec=3).f_rho_ZZ()
1 + 7/100*q + 21/160000*q^2 + O(q^3)
sage: MFSeriesConstructor(group=5, prec=3).f_rho_ZZ().parent()
Power Series Ring in q over Rational Field

sage: MFSeriesConstructor(group=infinity, prec=3).f_rho_ZZ()
1
```

group ( )

Return the (Hecke triangle) group of `self`.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.series_constructor import␣
↪MFSeriesConstructor
sage: MFSeriesConstructor(group=4).group()
Hecke triangle group for n = 4
```

hecke_n ( )

Return the parameter `n` of the (Hecke triangle) group of `self`.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.series_constructor import␣
↪MFSeriesConstructor
sage: MFSeriesConstructor(group=4).hecke_n()
4
```

prec ( )

Return the used default precision for the PowerSeriesRing or LaurentSeriesRing.

EXAMPLES:

```
sage: from sage.modular.modform_hecketriangle.series_constructor import␣
↪MFSeriesConstructor
sage: MFSeriesConstructor(group=5).prec()
10
sage: MFSeriesConstructor(group=5, prec=20).prec()
20
```

CHAPTER

FIFTEEN

INDICES AND TABLES

- Index
- Module Index
- Search Page