Databases

Release 10.6

The Sage Development Team

Jun 27, 2025

CONTENTS

Sage includes mathematical databases as standard or optional packages, and provides convenient interfaces for easy access to the databases.

CHAPTER

ONE

STANDARD DATABASES

These databases are available in the standard install of Sage or if Sage can access Internet.

1.1 Frank Lübeck's tables of Conway polynomials over finite fields

class sage.databases.conway.**ConwayPolynomials**

Bases: Mapping

Initialize the database.

degrees(*p*)

Return the list of integers `n` for which the database of Conway polynomials contains the polynomial of degree `n` over `GF(p)`.

EXAMPLES:

```
sage: c = ConwayPolynomials()
sage: c.degrees(60821)
[1, 2, 3, 4]
sage: c.degrees(next_prime(10^7))
[]
```

```
>>> from sage.all import *
>>> c = ConwayPolynomials()
>>> c.degrees(Integer(60821))
[1, 2, 3, 4]
>>> c.degrees(next_prime(Integer(10)**Integer(7)))
[]
```

has_polynomial(*p*, *n*)

Return `True` if the database of Conway polynomials contains the polynomial of degree `n` over `GF(p)`.

INPUT:

- `p` – prime number
- `n` – positive integer

EXAMPLES:

```
sage: c = ConwayPolynomials()
sage: c.has_polynomial(97, 12)
True
```

(continues on next page)

(continued from previous page)

```
sage: c.has_polynomial(60821, 5)
False
```

```
>>> from sage.all import *
>>> c = ConwayPolynomials()
>>> c.has_polynomial(Integer(97), Integer(12))
True
>>> c.has_polynomial(Integer(60821), Integer(5))
False
```

polynomial(*p*, *n*)

Return the Conway polynomial of degree `n` over `GF(p)`, or raise a `RuntimeError` if this polynomial is not in the database.

Note

See also the global function `conway_polynomial` for a more user-friendly way of accessing the polynomial.

INPUT:

- `p` – prime number
- `n` – positive integer

OUTPUT:

List of Python int's giving the coefficients of the corresponding Conway polynomial in ascending order of degree.

EXAMPLES:

```
sage: c = ConwayPolynomials()
sage: c.polynomial(3, 21)
(1, 2, 0, 2, 0, 1, 2, 0, 2, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1)
sage: c.polynomial(97, 128)
Traceback (most recent call last):
...
RuntimeError: Conway polynomial over F_97 of degree 128 not in database.
```

```
>>> from sage.all import *
>>> c = ConwayPolynomials()
>>> c.polynomial(Integer(3), Integer(21))
(1, 2, 0, 2, 0, 1, 2, 0, 2, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1)
>>> c.polynomial(Integer(97), Integer(128))
Traceback (most recent call last):
...
RuntimeError: Conway polynomial over F_97 of degree 128 not in database.
```

primes()

Return the list of prime numbers `p` for which the database of Conway polynomials contains polynomials over `GF(p)`.

EXAMPLES:

```
sage: c = ConwayPolynomials()
sage: P = c.primes()
sage: 2 in P
True
sage: next_prime(10^7) in P
False
```

```
>>> from sage.all import *
>>> c = ConwayPolynomials()
>>> P = c.primes()
>>> Integer(2) in P
True
>>> next_prime(Integer(10)**Integer(7)) in P
False
```

class sage.databases.conway.**DictInMapping**(*dict*)

Bases: Mapping

Places dict into a non-mutable mapping.

1.2 The On-Line Encyclopedia of Integer Sequences (OEIS)

You can query the OEIS (Online Database of Integer Sequences) through Sage in order to:

- identify a sequence from its first terms.
- obtain more terms, formulae, references, etc. for a given sequence.

EXAMPLES:

```
sage: oeis
The On-Line Encyclopedia of Integer Sequences (https://oeis.org/)
```

```
>>> from sage.all import *
>>> oeis
The On-Line Encyclopedia of Integer Sequences (https://oeis.org/)
```

What about a sequence starting with $3, 7, 15, 1$?

```
sage: # optional - internet
sage: search = oeis([3, 7, 15, 1], max_results=4); search   # random
0: A001203: Simple continued fraction expansion of Pi.
1: A240698: Partial sums of divisors of n, cf. A027750.
2: A082495: a(n) = (2^n - 1) mod n.
3: A165416: Irregular array read by rows: The n-th row contains those
            distinct positive integers that each, when written in binary,
            occurs as a substring in binary n.
sage: [u.id() for u in search]                              # random
['A001203', 'A240698', 'A082495', 'A165416']
sage: c = search[0]; c
A001203: Simple continued fraction expansion of Pi.
```

```
>>> from sage.all import *
>>> # optional - internet
>>> search = oeis([Integer(3), Integer(7), Integer(15), Integer(1)], max_
↪results=Integer(4)); search   # random
0: A001203: Simple continued fraction expansion of Pi.
1: A240698: Partial sums of divisors of n, cf. A027750.
2: A082495: a(n) = (2^n - 1) mod n.
3: A165416: Irregular array read by rows: The n-th row contains those
            distinct positive integers that each, when written in binary,
            occurs as a substring in binary n.
>>> [u.id() for u in search]                                    # random
['A001203', 'A240698', 'A082495', 'A165416']
>>> c = search[Integer(0)]; c
A001203: Simple continued fraction expansion of Pi.
```

```
sage: # optional - internet
sage: c.first_terms(15)
(3, 7, 15, 1, 292, 1, 1, 1, 2, 1, 3, 1, 14, 2, 1)
sage: c.examples()
0: Pi = 3.14159265358979323844...
1:    = 3 + 1/(7 + 1/(15 + 1/(1 + 1/(292 + ...))))
2:    = [a_0; a_1, a_2, a_3, ...] = [3; 7, 15, 1, 292, ...].
sage: c.comments()
0: The first 5821569425 terms were computed by _Eric W. Weisstein_ on Sep 18 2011.
1: The first 10672905501 terms were computed by _Eric W. Weisstein_ on Jul 17 2013.
2: The first 15000000000 terms were computed by _Eric W. Weisstein_ on Jul 27 2013.
3: The first 30113021586 terms were computed by _Syed Fahad_ on Apr 27 2021.
```

```
>>> from sage.all import *
>>> # optional - internet
>>> c.first_terms(Integer(15))
(3, 7, 15, 1, 292, 1, 1, 1, 2, 1, 3, 1, 14, 2, 1)
>>> c.examples()
0: Pi = 3.14159265358979323844...
1:    = 3 + 1/(7 + 1/(15 + 1/(1 + 1/(292 + ...))))
2:    = [a_0; a_1, a_2, a_3, ...] = [3; 7, 15, 1, 292, ...].
>>> c.comments()
0: The first 5821569425 terms were computed by _Eric W. Weisstein_ on Sep 18 2011.
1: The first 10672905501 terms were computed by _Eric W. Weisstein_ on Jul 17 2013.
2: The first 15000000000 terms were computed by _Eric W. Weisstein_ on Jul 27 2013.
3: The first 30113021586 terms were computed by _Syed Fahad_ on Apr 27 2021.
```

```
sage: # optional - internet
sage: x = c.natural_object(); type(x)
<class 'sage.rings.continued_fraction.ContinuedFraction_periodic'>
sage: x.convergents()[:7]
[3, 22/7, 333/106, 355/113, 103993/33102, 104348/33215, 208341/66317]
sage: RR(x.value())
3.14159265358979
sage: RR(x.value()) == RR(pi)
True
```

```
>>> from sage.all import *
>>> # optional - internet
>>> x = c.natural_object(); type(x)
<class 'sage.rings.continued_fraction.ContinuedFraction_periodic'>
>>> x.convergents()[:Integer(7)]
[3, 22/7, 333/106, 355/113, 103993/33102, 104348/33215, 208341/66317]
>>> RR(x.value())
3.14159265358979
>>> RR(x.value()) == RR(pi)
True
```

What about posets? Are they hard to count? To which other structures are they related?

```
sage: # optional - internet
sage: [Posets(i).cardinality() for i in range(10)]
[1, 1, 2, 5, 16, 63, 318, 2045, 16999, 183231]
sage: oeis(_)
0: A000112: Number of partially ordered sets ("posets") with n unlabeled elements.
sage: p = _[0]
sage: 'hard' in p.keywords()
True
sage: len(p.formulas())
0
sage: len(p.first_terms())
17
sage: p.cross_references(fetch=True)     # random
0: A000798: Number of different quasi-orders (or topologies, or transitive digraphs)
            with n labeled elements.
1: A001035: Number of partially ordered sets ("posets") with n labeled elements
            (or labeled acyclic transitive digraphs).
2: A001930: Number of topologies, or transitive digraphs with n unlabeled nodes.
3: A006057: Number of topologies on n labeled points satisfying axioms T_0-T_4.
4: A079263: Number of constrained mixed models with n factors.
5: A079265: Number of antisymmetric transitive binary relations on n unlabeled points.
6: A263859: Triangle read by rows: T(n,k) (n>=1, k>=0) is the number of posets
            with n elements and rank k (or depth k+1).
7: A316978: Number of factorizations of n into factors > 1 with no equivalent primes.
8: A319559: Number of non-isomorphic T_0 set systems of weight n.
9: A326939: Number of T_0 sets of subsets of {1..n} that cover all n vertices.
10: A326943: Number of T_0 sets of subsets of {1..n} that cover all n vertices and
             are closed under intersection.
...
```

```
>>> from sage.all import *
>>> # optional - internet
>>> [Posets(i).cardinality() for i in range(Integer(10))]
[1, 1, 2, 5, 16, 63, 318, 2045, 16999, 183231]
>>> oeis(_)
0: A000112: Number of partially ordered sets ("posets") with n unlabeled elements.
>>> p = _[Integer(0)]
>>> 'hard' in p.keywords()
True
>>> len(p.formulas())
```

(continues on next page)

(continued from previous page)

```
0
>>> len(p.first_terms())
17
>>> p.cross_references(fetch=True)     # random
0: A000798: Number of different quasi-orders (or topologies, or transitive digraphs)
            with n labeled elements.
1: A001035: Number of partially ordered sets ("posets") with n labeled elements
            (or labeled acyclic transitive digraphs).
2: A001930: Number of topologies, or transitive digraphs with n unlabeled nodes.
3: A006057: Number of topologies on n labeled points satisfying axioms T_0-T_4.
4: A079263: Number of constrained mixed models with n factors.
5: A079265: Number of antisymmetric transitive binary relations on n unlabeled points.
6: A263859: Triangle read by rows: T(n,k) (n>=1, k>=0) is the number of posets
            with n elements and rank k (or depth k+1).
7: A316978: Number of factorizations of n into factors > 1 with no equivalent primes.
8: A319559: Number of non-isomorphic T_0 set systems of weight n.
9: A326939: Number of T_0 sets of subsets of {1..n} that cover all n vertices.
10: A326943: Number of T_0 sets of subsets of {1..n} that cover all n vertices and
             are closed under intersection.
...
```

What does the Taylor expansion of the e^{e^x-1} function have to do with primes ?

```
sage: # optional - internet, needs sage.symbolic
sage: x = var('x') ; f(x) = e^(e^x - 1)
sage: L = [a*factorial(b) for a,b in taylor(f(x), x, 0, 20).coefficients()]; L
[1, 1, 2, 5, 15, 52, 203, 877, 4140, 21147, 115975, 678570, 4213597,
 27644437, 190899322, 1382958545, 10480142147, 82864869804, 682076806159,
 5832742205057, 51724158235372]
sage: oeis(L)
0: A000110: Bell or exponential numbers: number of ways to partition
            a set of n labeled elements.
1: A292935: E.g.f.: exp(exp(-x) - 1).
sage: b = _[0]
sage: b.formulas()[0]
'E.g.f.: exp(exp(x) - 1).'
sage: [i for i in b.comments() if 'prime' in i][-1]
'When n is prime, ...'
sage: [n for n in range(2, 20) if (b(n)-2) % n == 0]
[2, 3, 5, 7, 11, 13, 17, 19]
```

```
>>> from sage.all import *
>>> # optional - internet, needs sage.symbolic
>>> x = var('x') ; __tmp__=var("x"); f = symbolic_expression(e**(e**x - Integer(1))).
↪function(x)
>>> L = [a*factorial(b) for a,b in taylor(f(x), x, Integer(0), Integer(20)).
↪coefficients()]; L
[1, 1, 2, 5, 15, 52, 203, 877, 4140, 21147, 115975, 678570, 4213597,
 27644437, 190899322, 1382958545, 10480142147, 82864869804, 682076806159,
 5832742205057, 51724158235372]
>>> oeis(L)
0: A000110: Bell or exponential numbers: number of ways to partition
```

(continues on next page)

(continued from previous page)

```
            a set of n labeled elements.
1: A292935: E.g.f.: exp(exp(-x) - 1).
>>> b = _[Integer(0)]
>>> b.formulas()[Integer(0)]
'E.g.f.: exp(exp(x) - 1).'
>>> [i for i in b.comments() if 'prime' in i][-Integer(1)]
'When n is prime, ...'
>>> [n for n in range(Integer(2), Integer(20)) if (b(n)-Integer(2)) % n == Integer(0)]
[2, 3, 5, 7, 11, 13, 17, 19]
```

See also

- If you plan to do a lot of automatic searches for subsequences, you should consider installing *SloaneEncyclopedia*, a local partial copy of the OEIS.
- Some infinite OEIS sequences are implemented in Sage, via the `sloane_functions` module.

AUTHORS:

- Thierry Monteil (2012-02-10 – 2013-06-21): initial version.
- Vincent Delecroix (2014): modifies continued fractions because of Issue #14567
- Moritz Firsching (2016): modifies handling of dead sequence, see Issue #17330
- Thierry Monteil (2019): refactorization (unique representation Issue #28480, laziness Issue #28627)

class sage.databases.oeis.**FancyTuple**(*iterable=()*, */*)

Bases: `tuple`

This class inherits from `tuple`, it allows to nicely print tuples whose elements have a one line representation.

EXAMPLES:

```
sage: from sage.databases.oeis import FancyTuple
sage: t = FancyTuple(['zero', 'one', 'two', 'three', 4]); t
0: zero
1: one
2: two
3: three
4: 4

sage: t[2]
'two'
```

```
>>> from sage.all import *
>>> from sage.databases.oeis import FancyTuple
>>> t = FancyTuple(['zero', 'one', 'two', 'three', Integer(4)]); t
0: zero
1: one
2: two
3: three
4: 4
```

(continues on next page)

(continued from previous page)

```
>>> t[Integer(2)]
'two'
```

class sage.databases.oeis.**OEIS**

Bases: object

The On-Line Encyclopedia of Integer Sequences.

OEIS is a class representing the On-Line Encyclopedia of Integer Sequences. You can query it using its methods, but OEIS can also be called directly with three arguments:

- query – it can be:
  - a string representing an OEIS ID (e.g. 'A000045').
  - an integer representing an OEIS ID (e.g. 45).
  - a list representing a sequence of integers.
  - a string, representing a text search.
- max_results – (integer, default: 30) the maximum number of results to return, they are sorted according to their relevance. In any cases, the OEIS website will never provide more than 100 results.
- first_result – (integer, default: 0) allow to skip the first_result first results in the search, to go further. This is useful if you are looking for a sequence that may appear after the 100 first found sequences.

OUTPUT:

- if query is an integer or an OEIS ID (e.g. 'A000045'), returns the associated OEIS sequence.
- if query is a string, returns a tuple of OEIS sequences whose description corresponds to the query. Those sequences can be used without the need to fetch the database again.
- if query is a list or tuple of integers, returns a tuple of OEIS sequences containing it as a subsequence. Those sequences can be used without the need to fetch the database again.

EXAMPLES:

```
sage: oeis
The On-Line Encyclopedia of Integer Sequences (https://oeis.org/)
```

```
>>> from sage.all import *
>>> oeis
The On-Line Encyclopedia of Integer Sequences (https://oeis.org/)
```

A particular sequence can be called by its A-number or number:

```
sage: oeis('A000040')                              # optional -- internet
A000040: The prime numbers.
sage: oeis(45)                                     # optional -- internet
A000045: Fibonacci numbers: F(n) = F(n-1) + F(n-2) with F(0) = 0 and F(1) = 1.
```

```
>>> from sage.all import *
>>> oeis('A000040')                              # optional -- internet
A000040: The prime numbers.
>>> oeis(Integer(45))                                     # optional -- internet
A000045: Fibonacci numbers: F(n) = F(n-1) + F(n-2) with F(0) = 0 and F(1) = 1.
```

The database can be searched by subsequence:

```
sage: search = oeis([1,2,3,5,8,13]); search     # optional -- internet
0: A000045: Fibonacci numbers: F(n) = F(n-1) + F(n-2) with F(0) = 0 and F(1) = 1.
1: A290689: Number of transitive rooted trees with n nodes.
2: A027926: Triangular array T read by rows: T(n,0) = T(n,2n) = 1 for n >= 0;
            T(n,1) = 1 for n >= 1; T(n,k) = T(n-1,k-2) + T(n-1,k-1)
            for k = 2..2n-1, n >= 2.

sage: fibo = search[0]                          # optional -- internet

sage: fibo.name()                               # optional -- internet
'Fibonacci numbers: F(n) = F(n-1) + F(n-2) with F(0) = 0 and F(1) = 1.'

sage: print(fibo.first_terms())                 # optional -- internet
(0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597,
2584, 4181, 6765, 10946, 17711, 28657, 46368, 75025, 121393, 196418,
317811, 514229, 832040, 1346269, 2178309, 3524578, 5702887, 9227465,
14930352, 24157817, 39088169, 63245986, 102334155)

sage: fibo.cross_references()[0]                # optional -- internet
'A001622'

sage: fibo == oeis(45)                          # optional -- internet
True

sage: sfibo = oeis('A039834')                   # optional -- internet
sage: sfibo.first_terms()                       # optional -- internet
(1, 1, 0, 1, -1, 2, -3, 5, -8, 13, -21, 34, -55, 89, -144, 233,
-377, 610, -987, 1597, -2584, 4181, -6765, 10946, -17711, 28657,
-46368, 75025, -121393, 196418, -317811, 514229, -832040, 1346269,
-2178309, 3524578, -5702887, 9227465, -14930352, 24157817)

sage: tuple(abs(i) for i in sfibo.first_terms())[2:20] == fibo.first_terms()[:18]␣
↪  # optional -- internet
True

sage: fibo.formulas()[4]                        # optional -- internet
'F(n) = F(n-1) + F(n-2) = -(-1)^n F(-n).'

sage: fibo.comments()[6]                        # optional -- internet
"F(n+2) = number of binary sequences of length n that have no
consecutive 0's."

sage: fibo.links()[0]                           # optional -- internet
'https://oeis.org/A000045/b000045.txt'
```

```
>>> from sage.all import *
>>> search = oeis([Integer(1),Integer(2),Integer(3),Integer(5),Integer(8),
↪Integer(13)]); search     # optional -- internet
0: A000045: Fibonacci numbers: F(n) = F(n-1) + F(n-2) with F(0) = 0 and F(1) = 1.
1: A290689: Number of transitive rooted trees with n nodes.
2: A027926: Triangular array T read by rows: T(n,0) = T(n,2n) = 1 for n >= 0;
```

(continues on next page)

(continued from previous page)

```
            T(n,1) = 1 for n >= 1; T(n,k) = T(n-1,k-2) + T(n-1,k-1)
            for k = 2..2n-1, n >= 2.

>>> fibo = search[Integer(0)]                          # optional -- internet

>>> fibo.name()                                 # optional -- internet
'Fibonacci numbers: F(n) = F(n-1) + F(n-2) with F(0) = 0 and F(1) = 1.'

>>> print(fibo.first_terms())                   # optional -- internet
(0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597,
2584, 4181, 6765, 10946, 17711, 28657, 46368, 75025, 121393, 196418,
317811, 514229, 832040, 1346269, 2178309, 3524578, 5702887, 9227465,
14930352, 24157817, 39088169, 63245986, 102334155)

>>> fibo.cross_references()[Integer(0)]                # optional -- internet
'A001622'

>>> fibo == oeis(Integer(45))                          # optional -- internet
True

>>> sfibo = oeis('A039834')                    # optional -- internet
>>> sfibo.first_terms()                        # optional -- internet
(1, 1, 0, 1, -1, 2, -3, 5, -8, 13, -21, 34, -55, 89, -144, 233,
-377, 610, -987, 1597, -2584, 4181, -6765, 10946, -17711, 28657,
-46368, 75025, -121393, 196418, -317811, 514229, -832040, 1346269,
-2178309, 3524578, -5702887, 9227465, -14930352, 24157817)

>>> tuple(abs(i) for i in sfibo.first_terms())[Integer(2):Integer(20)] == fibo.
↪first_terms()[:Integer(18)]   # optional -- internet
True

>>> fibo.formulas()[Integer(4)]                        # optional -- internet
'F(n) = F(n-1) + F(n-2) = -(-1)^n F(-n).'

>>> fibo.comments()[Integer(6)]                        # optional -- internet
"F(n+2) = number of binary sequences of length n that have no
consecutive 0's."

>>> fibo.links()[Integer(0)]                           # optional -- internet
'https://oeis.org/A000045/b000045.txt'
```

The database can be searched by description:

```
sage: oeis('prime gap factorization', max_results=4)  # random  # optional --␣
↪internet
0: A073491: Numbers having no prime gaps in their factorization.
1: A073485: Product of any number of consecutive primes; squarefree numbers
            with no gaps in their prime factorization.
2: A073490: Number of prime gaps in factorization of n.
3: A073492: Numbers having at least one prime gap in their factorization.
```

```
>>> from sage.all import *
```

(continues on next page)

(continued from previous page)

```
>>> oeis('prime gap factorization', max_results=Integer(4))  # random  # optional␣
↪-- internet
0: A073491: Numbers having no prime gaps in their factorization.
1: A073485: Product of any number of consecutive primes; squarefree numbers
            with no gaps in their prime factorization.
2: A073490: Number of prime gaps in factorization of n.
3: A073492: Numbers having at least one prime gap in their factorization.
```

Note

The following will fetch the OEIS database only once:

```
sage: oeis([1,2,3,5,8,13])                    # optional -- internet
0: A000045: Fibonacci numbers: F(n) = F(n-1) + F(n-2) with F(0) = 0 and F(1) =␣
↪1.
1: A290689: Number of transitive rooted trees with n nodes.
2: A027926: Triangular array T read by rows: T(n,0) = T(n,2n) = 1 for n >= 0;
            T(n,1) = 1 for n >= 1; T(n,k) = T(n-1,k-2) + T(n-1,k-1)
            for k = 2..2n-1, n >= 2.

sage: oeis('A000045')                         # optional -- internet
A000045: Fibonacci numbers: F(n) = F(n-1) + F(n-2) with F(0) = 0 and F(1) = 1.
```

```
>>> from sage.all import *
>>> oeis([Integer(1),Integer(2),Integer(3),Integer(5),Integer(8),Integer(13)])␣
↪                 # optional -- internet
0: A000045: Fibonacci numbers: F(n) = F(n-1) + F(n-2) with F(0) = 0 and F(1) =␣
↪1.
1: A290689: Number of transitive rooted trees with n nodes.
2: A027926: Triangular array T read by rows: T(n,0) = T(n,2n) = 1 for n >= 0;
            T(n,1) = 1 for n >= 1; T(n,k) = T(n-1,k-2) + T(n-1,k-1)
            for k = 2..2n-1, n >= 2.

>>> oeis('A000045')                          # optional -- internet
A000045: Fibonacci numbers: F(n) = F(n-1) + F(n-2) with F(0) = 0 and F(1) = 1.
```

Indeed, due to some caching mechanism, the sequence is not re-created when called from its ID.

browse()

Open the OEIS web page in a browser.

EXAMPLES:

```
sage: oeis.browse()                           # optional -- webbrowser
```

```
>>> from sage.all import *
>>> oeis.browse()                             # optional -- webbrowser
```

find_by_description(*description*, *max_results=3*, *first_result=0*)

Search for OEIS sequences corresponding to the description.

INPUT:

- `description` – string; the description the searched sequences
- `max_results` – (integer, default: 3) the maximum number of results we want. In any case, the on-line encyclopedia will not return more than 100 results.
- `first_result` – (integer, default: 0) allow to skip the `first_result` first results in the search, to go further. This is useful if you are looking for a sequence that may appear after the 100 first found sequences.

OUTPUT:

- a tuple (with fancy formatting) of at most `max_results` OEIS sequences. Those sequences can be used without the need to fetch the database again.

EXAMPLES:

```
sage: oeis.find_by_description('prime gap factorization') # optional --␣
↪internet
0: A...: ...
1: A...: ...
2: A...: ...

sage: prime_gaps = _[2] ; prime_gaps         # optional -- internet
A...

sage: oeis('beaver')                         # optional -- internet
0: A...: ...eaver...
1: A...: ...eaver...
2: A...: ...eaver...

sage: oeis('beaver', max_results=4, first_result=2)     # optional -- internet
0: A...: ...eaver...
1: A...: ...eaver...
2: A...: ...eaver...
3: A...: ...eaver...
```

```
>>> from sage.all import *
>>> oeis.find_by_description('prime gap factorization') # optional -- internet
0: A...: ...
1: A...: ...
2: A...: ...

>>> prime_gaps = _[Integer(2)] ; prime_gaps         # optional -- internet
A...

>>> oeis('beaver')                          # optional -- internet
0: A...: ...eaver...
1: A...: ...eaver...
2: A...: ...eaver...

>>> oeis('beaver', max_results=Integer(4), first_result=Integer(2))     #␣
↪optional -- internet
0: A...: ...eaver...
1: A...: ...eaver...
2: A...: ...eaver...
3: A...: ...eaver...
```

find_by_entry(*entry*)

INPUT:

- `entry` – string corresponding to an entry in the internal format of the OEIS

OUTPUT: the corresponding OEIS sequence

EXAMPLES:

```
sage: entry = '%I A262002\n%N A262002 L.g.f.: log( Sum_{n>=0} x^n/n! *␣
↪Product_{k=1..n} (k^2 + 1) ).\n%K A262002 nonn'
sage: s = oeis.find_by_entry(entry)
sage: s
A262002: L.g.f.: log( Sum_{n>=0} x^n/n! * Product_{k=1..n} (k^2 + 1) ).
```

```
>>> from sage.all import *
>>> entry = '%I A262002\n%N A262002 L.g.f.: log( Sum_{n>=0} x^n/n! * Product_
↪{k=1..n} (k^2 + 1) ).\n%K A262002 nonn'
>>> s = oeis.find_by_entry(entry)
>>> s
A262002: L.g.f.: log( Sum_{n>=0} x^n/n! * Product_{k=1..n} (k^2 + 1) ).
```

find_by_id(*ident*, *fetch=False*)

INPUT:

- `ident` – string representing the A-number of the sequence or an integer representing its number
- `fetch` – boolean (default: `False`); whether to force fetching the content of the sequence on the internet

OUTPUT: the OEIS sequence whose A-number or number corresponds to `ident`

EXAMPLES:

```
sage: oeis.find_by_id('A000040')            # optional -- internet
A000040: The prime numbers.

sage: oeis.find_by_id(40)                   # optional -- internet
A000040: The prime numbers.
```

```
>>> from sage.all import *
>>> oeis.find_by_id('A000040')            # optional -- internet
A000040: The prime numbers.

>>> oeis.find_by_id(Integer(40))                  # optional -- internet
A000040: The prime numbers.
```

find_by_subsequence(*subsequence*, *max_results=3*, *first_result=0*)

Search for OEIS sequences containing the given subsequence.

INPUT:

- `subsequence` – list or tuple of integers
- `max_results` – integer (default: 3); the maximum of results requested
- `first_result` – integer (default: 0); allow to skip the `first_result` first results in the search, to go further. This is useful if you are looking for a sequence that may appear after the 100 first found sequences.

OUTPUT:

A tuple (with fancy formatting) of at most `max_results` OEIS sequences. Those sequences can be used without the need to fetch the database again.

EXAMPLES:

```
sage: oeis.find_by_subsequence([2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233,␣
↪377])  # optional -- internet # random
0: A000045: Fibonacci numbers: F(n) = F(n-1) + F(n-2) with F(0) = 0 and F(1)␣
↪= 1.
1: A212804: Expansion of (1 - x)/(1 - x - x^2).
2: A020695: Pisot sequence E(2,3).

sage: fibo = _[0] ; fibo                    # optional -- internet
A000045: Fibonacci numbers: F(n) = F(n-1) + F(n-2) with F(0) = 0 and F(1) = 1.
```

```
>>> from sage.all import *
>>> oeis.find_by_subsequence([Integer(2), Integer(3), Integer(5), Integer(8),␣
↪Integer(13), Integer(21), Integer(34), Integer(55), Integer(89),␣
↪Integer(144), Integer(233), Integer(377)])  # optional -- internet # random
0: A000045: Fibonacci numbers: F(n) = F(n-1) + F(n-2) with F(0) = 0 and F(1)␣
↪= 1.
1: A212804: Expansion of (1 - x)/(1 - x - x^2).
2: A020695: Pisot sequence E(2,3).

>>> fibo = _[Integer(0)] ; fibo                    # optional -- internet
A000045: Fibonacci numbers: F(n) = F(n-1) + F(n-2) with F(0) = 0 and F(1) = 1.
```

class sage.databases.oeis.**OEISSequence**(*ident*)

Bases: SageObject, UniqueRepresentation

The class of OEIS sequences.

This class implements OEIS sequences. They are usually produced by calls to the On-Line Encyclopedia of Integer Sequences, represented by the class *OEIS*.

Note

Since some sequences do not start with index 0, there is a difference between calling and getting item, see *__call__()* for more details

```
sage: # optional - internet
sage: sfibo = oeis('A039834')
sage: sfibo.first_terms()[:10]
(1, 1, 0, 1, -1, 2, -3, 5, -8, 13)
sage: sfibo(-2)
1
sage: sfibo(3)
2
sage: sfibo.offsets()
(-2, 6)
sage: sfibo[0]
1
sage: sfibo[6]
-3
```

```
>>> from sage.all import *
>>> # optional - internet
>>> sfibo = oeis('A039834')
>>> sfibo.first_terms()[:Integer(10)]
(1, 1, 0, 1, -1, 2, -3, 5, -8, 13)
>>> sfibo(-Integer(2))
1
>>> sfibo(Integer(3))
2
>>> sfibo.offsets()
(-2, 6)
>>> sfibo[Integer(0)]
1
>>> sfibo[Integer(6)]
-3
```

__call__(*k*)

Return the element of the sequence `self` with index `k`.

INPUT:

- `k` – integer

OUTPUT: integer

Note

The first index of the sequence `self` is not necessarily zero, it depends on the first offset of `self`. If the sequence represents the decimal expansion of a real number, the index 0 corresponds to the digit right after the decimal point.

EXAMPLES:

```
sage: f = oeis(45)                              # optional -- internet
sage: f.first_terms()[:10]                      # optional -- internet
(0, 1, 1, 2, 3, 5, 8, 13, 21, 34)

sage: f(4)                                      # optional -- internet
3
```

```
>>> from sage.all import *
>>> f = oeis(Integer(45))                               # optional -- internet
>>> f.first_terms()[:Integer(10)]                       # optional -- internet
(0, 1, 1, 2, 3, 5, 8, 13, 21, 34)

>>> f(Integer(4))                                       # optional -- internet
3
```

```
sage: sfibo = oeis('A039834')                   # optional -- internet
sage: sfibo.first_terms()[:10]                  # optional -- internet
(1, 1, 0, 1, -1, 2, -3, 5, -8, 13)
```

(continues on next page)

(continued from previous page)

```
sage: sfibo(-2)                                 # optional -- internet
1
sage: sfibo(4)                                  # optional -- internet
-3
sage: sfibo.offsets()                           # optional -- internet
(-2, 6)
```

```
>>> from sage.all import *
>>> sfibo = oeis('A039834')                  # optional -- internet
>>> sfibo.first_terms()[:Integer(10)]                # optional -- internet
(1, 1, 0, 1, -1, 2, -3, 5, -8, 13)

>>> sfibo(-Integer(2))                                # optional -- internet
1
>>> sfibo(Integer(4))                                 # optional -- internet
-3
>>> sfibo.offsets()                           # optional -- internet
(-2, 6)
```

author()

Return the author of the sequence in the encyclopedia.

OUTPUT: string

EXAMPLES:

```
sage: f = oeis(45) ; f                          # optional -- internet
A000045: Fibonacci numbers: F(n) = F(n-1) + F(n-2) with F(0) = 0 and F(1) = 1.

sage: f.author()                                # optional -- internet
'_N. J. A. Sloane_, 1964'
```

```
>>> from sage.all import *
>>> f = oeis(Integer(45)) ; f                         # optional -- internet
A000045: Fibonacci numbers: F(n) = F(n-1) + F(n-2) with F(0) = 0 and F(1) = 1.

>>> f.author()                                # optional -- internet
'_N. J. A. Sloane_, 1964'
```

browse()

Open the OEIS web page associated to the sequence `self` in a browser.

EXAMPLES:

```
sage: f = oeis(45); f                           # optional -- internet webbrowser
A000045: Fibonacci numbers: F(n) = F(n-1) + F(n-2) with F(0) = 0 and F(1) = 1.

sage: f.browse()                                # optional -- internet webbrowser
```

```
>>> from sage.all import *
>>> f = oeis(Integer(45)); f                          # optional -- internet␣
↪webbrowser
```

(continues on next page)

(continued from previous page)

```
A000045: Fibonacci numbers: F(n) = F(n-1) + F(n-2) with F(0) = 0 and F(1) = 1.

>>> f.browse()                             # optional -- internet webbrowser
```

comments()

Return a tuple of comments associated to the sequence `self`.

OUTPUT:

- tuple of strings (with fancy formatting).

EXAMPLES:

```
sage: f = oeis(45); f                          # optional -- internet
A000045: Fibonacci numbers: F(n) = F(n-1) + F(n-2) with F(0) = 0 and F(1) = 1.

sage: f.comments()[:8]                         # optional -- internet
0: D. E. Knuth writes...
1: In keeping with historical accounts...
2: Susantha Goonatilake writes...
3: Also sometimes called Hemachandra numbers.
4: Also sometimes called Lamé's sequence.
5: ...
6: F(n+2) = number of binary sequences of length n that have no consecutive 0
↪'s.
7: F(n+2) = number of subsets of {1,2,...,n} that contain no consecutive␣
↪integers.
```

```
>>> from sage.all import *
>>> f = oeis(Integer(45)); f                           # optional -- internet
A000045: Fibonacci numbers: F(n) = F(n-1) + F(n-2) with F(0) = 0 and F(1) = 1.

>>> f.comments()[:Integer(8)]                          # optional -- internet
0: D. E. Knuth writes...
1: In keeping with historical accounts...
2: Susantha Goonatilake writes...
3: Also sometimes called Hemachandra numbers.
4: Also sometimes called Lamé's sequence.
5: ...
6: F(n+2) = number of binary sequences of length n that have no consecutive 0
↪'s.
7: F(n+2) = number of subsets of {1,2,...,n} that contain no consecutive␣
↪integers.
```

cross_references(*fetch=False*)

Return a tuple of cross references associated to the sequence `self`.

INPUT:

- `fetch` – boolean (default: `False`)

OUTPUT:

- if `fetch` is `False`, return a list of OEIS IDs (strings).
- if `fetch` is `True`, return a tuple of OEIS sequences.

EXAMPLES:

```
sage: nbalanced = oeis("A005598") ; nbalanced   # optional -- internet
A005598: a(n) = 1 + Sum_{i=1..n} (n-i+1)*phi(i).

sage: nbalanced.cross_references()              # optional -- internet
('A000010', 'A002088', 'A011755', 'A049695', 'A049703', 'A103116')

sage: nbalanced.cross_references(fetch=True)    # optional -- internet
0: A000010: Euler totient function phi(n): count numbers <= n and prime to n.
1: A002088: Sum of totient function: a(n) = Sum_{k=1..n} phi(k), cf. A000010.
2: A011755: a(n) = Sum_{k=1..n} k*phi(k).
3: A049695: Array T read by diagonals; T(i,j) is the number of nonnegative
            slopes of lines determined by 2 lattice points in
            [ 0,i ] X [ 0,j ] if i > 0; T(0,j)=1 if j > 0; T(0,0)=0.
4: A049703: a(0) = 0; for n>0, a(n) = A005598(n)/2.
5: A103116: a(n) = Sum_{i=1..n} (n-i+1)*phi(i).

sage: phi = _[3]                                # optional -- internet
```

```
>>> from sage.all import *
>>> nbalanced = oeis("A005598") ; nbalanced   # optional -- internet
A005598: a(n) = 1 + Sum_{i=1..n} (n-i+1)*phi(i).

>>> nbalanced.cross_references()              # optional -- internet
('A000010', 'A002088', 'A011755', 'A049695', 'A049703', 'A103116')

>>> nbalanced.cross_references(fetch=True)    # optional -- internet
0: A000010: Euler totient function phi(n): count numbers <= n and prime to n.
1: A002088: Sum of totient function: a(n) = Sum_{k=1..n} phi(k), cf. A000010.
2: A011755: a(n) = Sum_{k=1..n} k*phi(k).
3: A049695: Array T read by diagonals; T(i,j) is the number of nonnegative
            slopes of lines determined by 2 lattice points in
            [ 0,i ] X [ 0,j ] if i > 0; T(0,j)=1 if j > 0; T(0,0)=0.
4: A049703: a(0) = 0; for n>0, a(n) = A005598(n)/2.
5: A103116: a(n) = Sum_{i=1..n} (n-i+1)*phi(i).

>>> phi = _[Integer(3)]                                # optional -- internet
```

examples()

Return a tuple of examples associated to the sequence `self`.

OUTPUT:

- tuple of strings (with fancy formatting).

EXAMPLES:

```
sage: c = oeis(1203); c                     # optional -- internet
A001203: Simple continued fraction expansion of Pi.

sage: c.examples()                          # optional -- internet
0: Pi = 3.14159265358979323...
1:    = 3 + 1/(7 + 1/(15 + 1/(1 + 1/(292 + ...))))
2:    = [a_0; a_1, a_2, a_3, ...] = [3; 7, 15, 1, 292, ...].
```

```
>>> from sage.all import *
>>> c = oeis(Integer(1203)); c                    # optional -- internet
A001203: Simple continued fraction expansion of Pi.

>>> c.examples()                              # optional -- internet
0: Pi = 3.1415926535897932384...
1:    = 3 + 1/(7 + 1/(15 + 1/(1 + 1/(292 + ...))))
2:    = [a_0; a_1, a_2, a_3, ...] = [3; 7, 15, 1, 292, ...].
```

extensions_or_errors()

Return a tuple of extensions or errors associated to the sequence `self`.

OUTPUT:

- tuple of strings (with fancy formatting).

EXAMPLES:

```
sage: sfibo = oeis('A039834'); sfibo        # optional -- internet
A039834: a(n+2) = -a(n+1) + a(n) (signed Fibonacci numbers) with
         a(-2) = a(-1) = 1; or Fibonacci numbers (A000045) extended
         to negative indices.

sage: sfibo.extensions_or_errors()[0]       # optional -- internet
'Signs corrected by _Len Smiley_ and _N. J. A. Sloane_'
```

```
>>> from sage.all import *
>>> sfibo = oeis('A039834'); sfibo        # optional -- internet
A039834: a(n+2) = -a(n+1) + a(n) (signed Fibonacci numbers) with
         a(-2) = a(-1) = 1; or Fibonacci numbers (A000045) extended
         to negative indices.

>>> sfibo.extensions_or_errors()[Integer(0)]       # optional -- internet
'Signs corrected by _Len Smiley_ and _N. J. A. Sloane_'
```

first_terms(*number=None*)

INPUT:

- `number` – integer or `None` (default); the number of terms returned (if less than the number of available terms). When set to `None`, returns all the known terms.

OUTPUT: tuple of integers

EXAMPLES:

```
sage: f = oeis(45); f                       # optional -- internet
A000045: Fibonacci numbers: F(n) = F(n-1) + F(n-2) with F(0) = 0 and F(1) = 1.

sage: f.first_terms()[:10]                  # optional -- internet
(0, 1, 1, 2, 3, 5, 8, 13, 21, 34)
```

```
>>> from sage.all import *
>>> f = oeis(Integer(45)); f                       # optional -- internet
A000045: Fibonacci numbers: F(n) = F(n-1) + F(n-2) with F(0) = 0 and F(1) = 1.
```

(continues on next page)

(continued from previous page)

```
>>> f.first_terms()[:Integer(10)]                    # optional -- internet
(0, 1, 1, 2, 3, 5, 8, 13, 21, 34)
```

Handle dead sequences, see Issue #17330

```
sage: oeis(5000).first_terms(12)             # optional -- internet
(1, 0, 0, 1, 1, 1, 11, 36, 92, 491, 2537)
```

```
>>> from sage.all import *
>>> oeis(Integer(5000)).first_terms(Integer(12))            # optional --␣
↪internet
(1, 0, 0, 1, 1, 1, 11, 36, 92, 491, 2537)
```

formulas()

Return a tuple of formulas associated to the sequence `self`.

OUTPUT:

- tuple of strings (with fancy formatting).

EXAMPLES:

```
sage: f = oeis(45) ; f                       # optional -- internet
A000045: Fibonacci numbers: F(n) = F(n-1) + F(n-2) with F(0) = 0 and F(1) = 1.

sage: f.formulas()[2]                        # optional -- internet
'F(n) = ((1+sqrt(5))^n - (1-sqrt(5))^n)/(2^n*sqrt(5)).'
```

```
>>> from sage.all import *
>>> f = oeis(Integer(45)) ; f                       # optional -- internet
A000045: Fibonacci numbers: F(n) = F(n-1) + F(n-2) with F(0) = 0 and F(1) = 1.

>>> f.formulas()[Integer(2)]                        # optional -- internet
'F(n) = ((1+sqrt(5))^n - (1-sqrt(5))^n)/(2^n*sqrt(5)).'
```

id(*format='A'*)

The ID of the sequence `self` is the A-number that identifies `self`.

INPUT:

- `format` – string (default: `'A'`)

OUTPUT:

- if `format` is set to 'A', returns a string of the form 'A000123'.
- if `format` is set to 'int' returns an integer of the form 123.

EXAMPLES:

```
sage: f = oeis(45) ; f                       # optional -- internet
A000045: Fibonacci numbers: F(n) = F(n-1) + F(n-2) with F(0) = 0 and F(1) = 1.

sage: f.id()                                 # optional -- internet
'A000045'
```

(continues on next page)

(continued from previous page)

```
sage: f.id(format='int')                    # optional -- internet
45
```

```
>>> from sage.all import *
>>> f = oeis(Integer(45)) ; f                    # optional -- internet
A000045: Fibonacci numbers: F(n) = F(n-1) + F(n-2) with F(0) = 0 and F(1) = 1.

>>> f.id()                                  # optional -- internet
'A000045'

>>> f.id(format='int')                      # optional -- internet
45
```

is_dead(*warn_only=False*)

Tell whether the sequence is dead.

INPUT:

- `warn_only` – ignored

EXAMPLES:

sage: s = oeis(17) # optional – internet sage: s # optional – internet A000017: Erroneous version of A032522.

is_finite()

Tell whether the sequence is finite.

Currently, OEIS only provides a keyword when the sequence is known to be finite. So, when this keyword is not there, we do not know whether it is infinite or not.

OUTPUT:

- `True` when the sequence is known to be finite.
- `Unknown` otherwise.

Todo

Ask OEIS for a keyword ensuring that a sequence is infinite.

EXAMPLES:

```
sage: s = oeis('A114288') ; s               # optional -- internet
A114288: Lexicographically earliest solution of any 9 X 9 sudoku, read by␣
↪rows.

sage: s.is_finite()                         # optional -- internet
True
```

```
>>> from sage.all import *
>>> s = oeis('A114288') ; s               # optional -- internet
A114288: Lexicographically earliest solution of any 9 X 9 sudoku, read by␣
↪rows.
```

(continues on next page)

(continued from previous page)

```
>>> s.is_finite()                          # optional -- internet
True
```

```
sage: f = oeis(45) ; f                       # optional -- internet
A000045: Fibonacci numbers: F(n) = F(n-1) + F(n-2) with F(0) = 0 and F(1) = 1.

sage: f.is_finite()                          # optional -- internet
Unknown
```

```
>>> from sage.all import *
>>> f = oeis(Integer(45)) ; f                       # optional -- internet
A000045: Fibonacci numbers: F(n) = F(n-1) + F(n-2) with F(0) = 0 and F(1) = 1.

>>> f.is_finite()                          # optional -- internet
Unknown
```

is_full()

Tell whether the sequence `self` is full, that is, if all its elements are listed in `self.first_terms()`.

Currently, OEIS only provides a keyword when the sequence is known to be full. So, when this keyword is not there, we do not know whether some elements are missing or not.

OUTPUT:

- `True` when the sequence is known to be full.
- `Unknown` otherwise.

EXAMPLES:

```
sage: s = oeis('A114288') ; s                # optional -- internet
A114288: Lexicographically earliest solution of any 9 X 9 sudoku, read by␣
↪rows.

sage: s.is_full()                            # optional -- internet
True
```

```
>>> from sage.all import *
>>> s = oeis('A114288') ; s                # optional -- internet
A114288: Lexicographically earliest solution of any 9 X 9 sudoku, read by␣
↪rows.

>>> s.is_full()                             # optional -- internet
True
```

```
sage: f = oeis(45) ; f                       # optional -- internet
A000045: Fibonacci numbers: F(n) = F(n-1) + F(n-2) with F(0) = 0 and F(1) = 1.

sage: f.is_full()                            # optional -- internet
Unknown
```

```
>>> from sage.all import *
>>> f = oeis(Integer(45)) ; f                       # optional -- internet
```

(continues on next page)

(continued from previous page)

```
A000045: Fibonacci numbers: F(n) = F(n-1) + F(n-2) with F(0) = 0 and F(1) = 1.

>>> f.is_full()                              # optional -- internet
Unknown
```

keywords(*warn=True*)

Return the keywords associated to the sequence `self`.

OUTPUT: tuple of strings

EXAMPLES:

```
sage: f = oeis(53) ; f                       # optional -- internet
A000053: Local stops on New York City...

sage: f.keywords()                           # optional -- internet
('nonn', 'fini', ...)
```

```
>>> from sage.all import *
>>> f = oeis(Integer(53)) ; f                       # optional -- internet
A000053: Local stops on New York City...

>>> f.keywords()                             # optional -- internet
('nonn', 'fini', ...)
```

links(*browse=None*, *format='guess'*)

Return, display or browse links associated to the sequence `self`.

INPUT:

- `browse` – integer; a list of integers, or the word 'all' (default: `None`) which links to open in a web browser
- `format` – string (default: `'guess'`); how to display the links

OUTPUT: tuple of strings (with fancy formatting):

- if `format` is `url`, returns a tuple of absolute links without description.
- if `format` is `html`, returns nothing but prints a tuple of clickable absolute links in their context.
- if `format` is `guess`, adapts the output to the context (command line or notebook).
- if `format` is `raw`, the links as they appear in the database, relative links are not made absolute.

EXAMPLES:

```
sage: f = oeis(45) ; f                       # optional -- internet
A000045: Fibonacci numbers: F(n) = F(n-1) + F(n-2) with F(0) = 0 and F(1) = 1.

sage: f.links(format='url')                              # optional -- internet
0: https://oeis.org/A000045/b000045.txt
1: ...
2: ...

sage: f.links(format='raw')                  # optional -- internet
0: N. J. A. Sloane, <a href="/A000045/b000045.txt">The first 2000 Fibonacci␣
↪numbers: Table of n, F(n) for n = 0..2000</a>
```

(continues on next page)

(continued from previous page)

```
1: ...
2: ...
```

```
>>> from sage.all import *
>>> f = oeis(Integer(45)) ; f                          # optional -- internet
A000045: Fibonacci numbers: F(n) = F(n-1) + F(n-2) with F(0) = 0 and F(1) = 1.

>>> f.links(format='url')                               # optional -- internet
0: https://oeis.org/A000045/b000045.txt
1: ...
2: ...

>>> f.links(format='raw')                  # optional -- internet
0: N. J. A. Sloane, <a href="/A000045/b000045.txt">The first 2000 Fibonacci␣
↪numbers: Table of n, F(n) for n = 0..2000</a>
1: ...
2: ...
```

name()

Return the name of the sequence `self`.

OUTPUT: string

EXAMPLES:

```
sage: f = oeis(45) ; f                            # optional -- internet
A000045: Fibonacci numbers: F(n) = F(n-1) + F(n-2) with F(0) = 0 and F(1) = 1.

sage: f.name()                                    # optional -- internet
'Fibonacci numbers: F(n) = F(n-1) + F(n-2) with F(0) = 0 and F(1) = 1.'
```

```
>>> from sage.all import *
>>> f = oeis(Integer(45)) ; f                          # optional -- internet
A000045: Fibonacci numbers: F(n) = F(n-1) + F(n-2) with F(0) = 0 and F(1) = 1.

>>> f.name()                                 # optional -- internet
'Fibonacci numbers: F(n) = F(n-1) + F(n-2) with F(0) = 0 and F(1) = 1.'
```

natural_object()

Return the natural object associated to the sequence `self`.

OUTPUT:

- If the sequence `self` corresponds to the digits of a real number, returns the associated real number (as an element of RealLazyField()).
- If the sequence `self` corresponds to the convergents of a continued fraction, returns the associated continued fraction.

Warning

This method forgets the fact that the returned sequence may not be complete.

Todo

- ask OEIS to add a keyword telling whether the sequence comes from a power series, e.g. for OEIS sequence A000182
- discover other possible conversions.

EXAMPLES:

```
sage: g = oeis("A002852"); g                # optional -- internet
A002852: Continued fraction for Euler's constant (or Euler-Mascheroni␣
↪constant) gamma.

sage: x = g.natural_object(); type(x)       # optional -- internet
<class 'sage.rings.continued_fraction.ContinuedFraction_periodic'>

sage: RDF(x) == RDF(euler_gamma)            # optional -- internet
True

sage: cfg = continued_fraction(euler_gamma)                                  #␣
↪needs sage.symbolic
sage: x[:90] == cfg[:90]                    # optional - internet              #␣
↪needs sage.symbolic
True
```

```
>>> from sage.all import *
>>> g = oeis("A002852"); g                # optional -- internet
A002852: Continued fraction for Euler's constant (or Euler-Mascheroni␣
↪constant) gamma.

>>> x = g.natural_object(); type(x)       # optional -- internet
<class 'sage.rings.continued_fraction.ContinuedFraction_periodic'>

>>> RDF(x) == RDF(euler_gamma)            # optional -- internet
True

>>> cfg = continued_fraction(euler_gamma)                                    #␣
↪needs sage.symbolic
>>> x[:Integer(90)] == cfg[:Integer(90)]                    # optional -␣
↪internet           # needs sage.symbolic
True
```

```
sage: ee = oeis('A001113'); ee              # optional -- internet
A001113: Decimal expansion of e.

sage: x = ee.natural_object(); x            # optional -- internet
2.718281828459046?

sage: x.parent()                            # optional -- internet
Real Lazy Field

sage: x == RR(e)                            # optional -- internet
```

(continues on next page)

(continued from previous page)

```
True
```

```
>>> from sage.all import *
>>> ee = oeis('A001113'); ee              # optional -- internet
A001113: Decimal expansion of e.

>>> x = ee.natural_object(); x            # optional -- internet
2.718281828459046?

>>> x.parent()                            # optional -- internet
Real Lazy Field

>>> x == RR(e)                            # optional -- internet
True
```

```
sage: av = oeis('A322578'); av              # optional -- internet
A322578: Decimal expansion ... Avogadro...

sage: av.natural_object()                   # optional -- internet
6.022140760000000?e23
```

```
>>> from sage.all import *
>>> av = oeis('A322578'); av              # optional -- internet
A322578: Decimal expansion ... Avogadro...

>>> av.natural_object()                   # optional -- internet
6.022140760000000?e23
```

```
sage: fib = oeis('A000045'); fib            # optional -- internet
A000045: Fibonacci numbers: F(n) = F(n-1) + F(n-2) with F(0) = 0 and F(1) = 1.

sage: x = fib.natural_object(); x.universe()    # optional -- internet
Non negative integer semiring
```

```
>>> from sage.all import *
>>> fib = oeis('A000045'); fib            # optional -- internet
A000045: Fibonacci numbers: F(n) = F(n-1) + F(n-2) with F(0) = 0 and F(1) = 1.

>>> x = fib.natural_object(); x.universe()    # optional -- internet
Non negative integer semiring
```

```
sage: sfib = oeis('A039834'); sfib         # optional -- internet
A039834: a(n+2) = -a(n+1) + a(n) (signed Fibonacci numbers) with
         a(-2) = a(-1) = 1; or Fibonacci numbers (A000045)
         extended to negative indices.

sage: x = sfib.natural_object(); x.universe()   # optional -- internet
Integer Ring
```

```
>>> from sage.all import *
```

(continues on next page)

(continued from previous page)

```
>>> sfib = oeis('A039834'); sfib         # optional -- internet
A039834: a(n+2) = -a(n+1) + a(n) (signed Fibonacci numbers) with
         a(-2) = a(-1) = 1; or Fibonacci numbers (A000045)
         extended to negative indices.

>>> x = sfib.natural_object(); x.universe()    # optional -- internet
Integer Ring
```

offsets()

Return the offsets of the sequence `self`.

The first offset is the subscript of the first term in the sequence `self`. When, the sequence represents the decimal expansion of a real number, it corresponds to the number of digits of its integer part.

The second offset is the first term in the sequence `self` (starting from 1) whose absolute value is greater than 1. This is set to 1 if all the terms are 0 or +-1.

OUTPUT: tuple of two elements

EXAMPLES:

```
sage: f = oeis(45) ; f                        # optional -- internet
A000045: Fibonacci numbers: F(n) = F(n-1) + F(n-2) with F(0) = 0 and F(1) = 1.

sage: f.offsets()                             # optional -- internet
(0, 4)

sage: f.first_terms()[:4]                     # optional -- internet
(0, 1, 1, 2)
```

```
>>> from sage.all import *
>>> f = oeis(Integer(45)) ; f                        # optional -- internet
A000045: Fibonacci numbers: F(n) = F(n-1) + F(n-2) with F(0) = 0 and F(1) = 1.

>>> f.offsets()                               # optional -- internet
(0, 4)

>>> f.first_terms()[:Integer(4)]                     # optional -- internet
(0, 1, 1, 2)
```

old_IDs()

Return the IDs of the sequence `self` corresponding to ancestors of OEIS.

OUTPUT:

- a tuple of at most two strings. When the string starts with M, it corresponds to the ID of "The Encyclopedia of Integer Sequences" of 1995. When the string starts with N, it corresponds to the ID of the "Handbook of Integer Sequences" of 1973.

EXAMPLES:

```
sage: f = oeis(45) ; f                        # optional -- internet
A000045: Fibonacci numbers: F(n) = F(n-1) + F(n-2) with F(0) = 0 and F(1) = 1.

sage: f.old_IDs()                             # optional -- internet
('M0692', 'N0256')
```

```
>>> from sage.all import *
>>> f = oeis(Integer(45)) ; f                          # optional -- internet
A000045: Fibonacci numbers: F(n) = F(n-1) + F(n-2) with F(0) = 0 and F(1) = 1.

>>> f.old_IDs()                                 # optional -- internet
('M0692', 'N0256')
```

online_update()

Fetch the online OEIS to update the informations about this sequence.

programs(*language='all'*, *preparsing=True*, *keep_comments=False*)

Return programs for the sequence `self` in the given `language`.

INPUT:

- `language` – string (default: `'all'`); the chosen language. Possible values are 'all' for the full list, or any language name, for example 'sage', 'maple', 'mathematica', etc.

Some further optional input is specific to sage code treatment:

- `preparsing` – boolean (default: `True`); whether to preparse sage code
- `keep_comments` – boolean (default: `False`); whether to keep comments in sage code

OUTPUT:

If `language` is `'all'`, this returns a sorted list of pairs (language, code), where every language can appear several times.

Otherwise, this returns a list of programs in the `language`, each program being a tuple of strings (with fancy formatting).

EXAMPLES:

```
sage: ee = oeis.find_by_id('A00260')         # optional -- internet
sage: ee.programs('pari')[0]                 # optional -- internet
0: {a(n) = binomial(...)};...

sage: G = oeis.find_by_id('A27642')    # optional -- internet
sage: G.programs('all')                # optional -- internet
[('haskell', ...),
 ('magma', ...),
 ...
 ('python', ...),
 ('sage', ...)]
```

```
>>> from sage.all import *
>>> ee = oeis.find_by_id('A00260')         # optional -- internet
>>> ee.programs('pari')[Integer(0)]                 # optional -- internet
0: {a(n) = binomial(...)};...

>>> G = oeis.find_by_id('A27642')    # optional -- internet
>>> G.programs('all')                # optional -- internet
[('haskell', ...),
 ('magma', ...),
 ...
```

(continues on next page)

(continued from previous page)

```
 ('python', ...),
 ('sage', ...)]
```

raw_entry()

Return the raw entry of the sequence `self`, in the OEIS format.

The raw entry is fetched online if needed.

OUTPUT: string

EXAMPLES:

```
sage: f = oeis(45) ; f                          # optional -- internet
A000045: Fibonacci numbers: F(n) = F(n-1) + F(n-2) with F(0) = 0 and F(1) = 1.

sage: print(f.raw_entry())                      # optional -- internet
%I A000045 M0692 N0256...
%S A000045 0,1,1,2,3,5,8,13,21,34,55,89,144,...
%T A000045 10946,17711,28657,46368,...
...
```

```
>>> from sage.all import *
>>> f = oeis(Integer(45)) ; f                          # optional -- internet
A000045: Fibonacci numbers: F(n) = F(n-1) + F(n-2) with F(0) = 0 and F(1) = 1.

>>> print(f.raw_entry())                      # optional -- internet
%I A000045 M0692 N0256...
%S A000045 0,1,1,2,3,5,8,13,21,34,55,89,144,...
%T A000045 10946,17711,28657,46368,...
...
```

references()

Return a tuple of references associated to the sequence `self`.

OUTPUT:

- tuple of strings (with fancy formatting).

EXAMPLES:

```
sage: w = oeis(7540) ; w                        # optional -- internet
A007540: Wilson primes: primes p such that (p-1)! == -1 (mod p^2).

sage: w.references()                            # optional -- internet
...Albert H. Beiler, Recreations in the Theory of Numbers, Dover, NY, 1964, p.
↪ 52.
...
```

```
>>> from sage.all import *
>>> w = oeis(Integer(7540)) ; w                        # optional -- internet
A007540: Wilson primes: primes p such that (p-1)! == -1 (mod p^2).

>>> w.references()                            # optional -- internet
...Albert H. Beiler, Recreations in the Theory of Numbers, Dover, NY, 1964, p.
```

(continues on next page)

(continued from previous page)

```
↪ 52.
...
```

show()

Display most available information about the sequence `self`.

EXAMPLES:

```
sage: s = oeis(12345)                          # optional -- internet
sage: s.show()                                 # optional -- internet
ID
A012345

NAME
Coefficients in the expansion sinh(arcsin(x)*arcsin(x)) = 2*x^2/2!+8*x^4/4!
↪+248*x^6/6!+11328*x^8/8!+...

FIRST TERMS
(2, 8, 248, 11328, 849312, 94857600, 14819214720, 3091936512000,␣
↪831657655349760, 280473756197529600, 115967597965430077440,␣
↪5771225789245691912960, 34039765801079493369569280)

LINKS
0: https://oeis.org/A012345/b012345.txt

FORMULAS
...
OFFSETS
(0, 1)

URL
https://oeis.org/A012345

AUTHOR
Patrick Demichel (patrick.demichel(AT)hp.com)
```

```
>>> from sage.all import *
>>> s = oeis(Integer(12345))                          # optional -- internet
>>> s.show()                                  # optional -- internet
ID
A012345
<BLANKLINE>
NAME
Coefficients in the expansion sinh(arcsin(x)*arcsin(x)) = 2*x^2/2!+8*x^4/4!
↪+248*x^6/6!+11328*x^8/8!+...
<BLANKLINE>
FIRST TERMS
(2, 8, 248, 11328, 849312, 94857600, 14819214720, 3091936512000,␣
↪831657655349760, 280473756197529600, 115967597965430077440,␣
↪5771225789245691912960, 34039765801079493369569280)
<BLANKLINE>
LINKS
```

(continues on next page)

(continued from previous page)

```
0: https://oeis.org/A012345/b012345.txt
<BLANKLINE>
FORMULAS
...
OFFSETS
(0, 1)
<BLANKLINE>
URL
https://oeis.org/A012345
<BLANKLINE>
AUTHOR
Patrick Demichel (patrick.demichel(AT)hp.com)
<BLANKLINE>
```

test_compile_sage_code()

Try to compile the extracted sage code, if there is any.

If there are several sage code fields, they are all considered.

Dead sequences are considered to compile correctly by default.

This returns `True` if the code compiles, and raises an error otherwise.

EXAMPLES:

One correct sequence:

```
sage: s = oeis.find_by_id('A027642')         # optional -- internet
sage: s.test_compile_sage_code()             # optional -- internet
True
```

```
>>> from sage.all import *
>>> s = oeis.find_by_id('A027642')         # optional -- internet
>>> s.test_compile_sage_code()             # optional -- internet
True
```

One dead sequence:

```
sage: s = oeis.find_by_id('A000154')         # optional -- internet
sage: s.test_compile_sage_code()             # optional -- internet
True
```

```
>>> from sage.all import *
>>> s = oeis.find_by_id('A000154')         # optional -- internet
>>> s.test_compile_sage_code()             # optional -- internet
True
```

url()

Return the URL of the page associated to the sequence `self`.

OUTPUT: string

EXAMPLES:

```
sage: f = oeis(45); f                          # optional -- internet
A000045: Fibonacci numbers: F(n) = F(n-1) + F(n-2) with F(0) = 0 and F(1) = 1.

sage: f.url()                                  # optional -- internet
'https://oeis.org/A000045'
```

```
>>> from sage.all import *
>>> f = oeis(Integer(45)); f                          # optional -- internet
A000045: Fibonacci numbers: F(n) = F(n-1) + F(n-2) with F(0) = 0 and F(1) = 1.

>>> f.url()                                  # optional -- internet
'https://oeis.org/A000045'
```

sage.databases.oeis.**to_tuple**(*string*)

Convert a string to a tuple of integers.

EXAMPLES:

```
sage: from sage.databases.oeis import to_tuple
sage: to_tuple('12,55,273')
(12, 55, 273)
```

```
>>> from sage.all import *
>>> from sage.databases.oeis import to_tuple
>>> to_tuple('12,55,273')
(12, 55, 273)
```

1.3 Local copy of the On-Line Encyclopedia of Integer Sequences

The `SloaneEncyclopedia` object provides access to a local copy of the database containing only the sequences and their names. To use this, you must download and install the database using `SloaneEncyclopedia.install()`, or `SloaneEncyclopedia.install_from_gz()` if you have already downloaded the database manually.

To look up a sequence, type

```
sage: SloaneEncyclopedia[60843]  # optional - sloane_database
[1, 6, 21, 107, 47176870]
```

```
>>> from sage.all import *
>>> SloaneEncyclopedia[Integer(60843)]  # optional - sloane_database
[1, 6, 21, 107, 47176870]
```

To get the name of a sequence, type

```
sage: SloaneEncyclopedia.sequence_name(1)  # optional - sloane_database
'Number of groups of order n.'
```

```
>>> from sage.all import *
>>> SloaneEncyclopedia.sequence_name(Integer(1))  # optional - sloane_database
'Number of groups of order n.'
```

To search locally for a particular subsequence, type

```
sage: SloaneEncyclopedia.find([1,2,3,4,5], 1)     # optional - sloane_database
[(15, [1, 2, 3, 4, 5, 7, 7, 8, 9, 11, 11, 13, 13, 16, 16, 16, 17, 19, 19, 23, 23, 23,␣
↪23, 25, 25, 27, 27, 29, 29, 31, 31, 32, 37, 37, 37, 37, 37, 41, 41, 41, 41, 43, 43,␣
↪47, 47, 47, 47, 49, 49, 53, 53, 53, 53, 59, 59, 59, 59, 59, 59, 61, 61, 64, 64, 64,␣
↪67, 67, 67, 71, 71, 71, 71, 73])]
```

```
>>> from sage.all import *
>>> SloaneEncyclopedia.find([Integer(1),Integer(2),Integer(3),Integer(4),Integer(5)],␣
↪Integer(1))     # optional - sloane_database
[(15, [1, 2, 3, 4, 5, 7, 7, 8, 9, 11, 11, 13, 13, 16, 16, 16, 17, 19, 19, 23, 23, 23,␣
↪23, 25, 25, 27, 27, 29, 29, 31, 31, 32, 37, 37, 37, 37, 37, 41, 41, 41, 41, 43, 43,␣
↪47, 47, 47, 47, 49, 49, 53, 53, 53, 53, 59, 59, 59, 59, 59, 59, 61, 61, 64, 64, 64,␣
↪67, 67, 67, 71, 71, 71, 71, 73])]
```

The default maximum number of results is 30, but to return up to 100, type

```
sage: SloaneEncyclopedia.find([1,2,3,4,5], 100)  # optional - sloane_database
[(15, [1, 2, 3, 4, 5, 7, 7, 8, 9, 11, 11, ...
```

```
>>> from sage.all import *
>>> SloaneEncyclopedia.find([Integer(1),Integer(2),Integer(3),Integer(4),Integer(5)],␣
↪Integer(100))  # optional - sloane_database
[(15, [1, 2, 3, 4, 5, 7, 7, 8, 9, 11, 11, ...
```

Results in either case are of the form [(number, list)].

See also

- If you want to get more informations relative to a sequence (references, links, examples, programs, …), you can use the On-Line Encyclopedia of Integer Sequences provided by the *OEIS* module.
- Some infinite OEIS sequences are implemented in Sage, via the `sloane_functions` module.

AUTHORS:

- Steven Sivek (2005-12-22): first version
- Steven Sivek (2006-02-07): updated to correctly handle the new search form on the Sloane website, and it is now also smarter about loading the local database in that it does not convert a sequence from string form to a list of integers until absolutely necessary. This seems to cut the loading time roughly in half.
- Steven Sivek (2009-12-22): added the SloaneEncyclopedia functions install() and install_from_gz() so users can get the latest versions of the OEIS without having to get an updated spkg; added sequence_name() to return the description of a sequence; and changed the data type for elements of each sequence from int to Integer.
- Thierry Monteil (2012-02-10): deprecated dead code and update related doc and tests.

class sage.databases.sloane.**SloaneEncyclopediaClass**

Bases: `object`

A local copy of the Sloane Online Encyclopedia of Integer Sequences that contains only the sequence numbers and the sequences themselves.

find(*seq*, *maxresults=30*)

Return a list of all sequences which have seq as a subsequence, up to maxresults results. Sequences are returned in the form (number, list).

INPUT:

- `seq` – list
- `maxresults` – integer

OUTPUT: list of 2-tuples (i, v), where v is a sequence with seq as a subsequence.

install(*oeis_url='https://oeis.org/stripped.gz'*, *names_url='https://oeis.org/names.gz'*, *overwrite=False*)

Download and install the online encyclopedia, raising an IOError if either step fails.

INPUT:

- `oeis_url` – string (default: `'https://oeis.org...'`) The URL of the stripped.gz encyclopedia file
- `names_url` – string (default: `'https://oeis.org...'`) The URL of the names.gz encyclopedia file. If you do not want to download this file, set names_url=None.
- `overwrite` – boolean (default: `False`); if the encyclopedia is already installed and overwrite=True, download and install the latest version over the installed one

install_from_gz(*stripped_file*, *names_file*, *overwrite=False*)

Install the online encyclopedia from a local stripped.gz file.

INPUT:

- `stripped_file` – string; the name of the stripped.gz OEIS file
- `names_file` – string; the name of the names.gz OEIS file, or None if the user does not want it installed
- `overwrite` – boolean (default: `False`); if the encyclopedia is already installed and `overwrite=True`, install 'filename' over the old encyclopedia

is_installed()

Check if a local copy of the encyclopedia is installed.

EXAMPLES:

```
sage: SloaneEncyclopedia.is_installed()  # optional - sloane_database
True
```

```
>>> from sage.all import *
>>> SloaneEncyclopedia.is_installed()  # optional - sloane_database
True
```

load()

Load the entire encyclopedia into memory from a file. This is done automatically if the user tries to perform a lookup or a search.

sequence_name(*N*)

Return the name of sequence `N` in the encyclopedia.

If sequence `N` does not exist, return `''`. If the names database is not installed, raise an `IOError`.

INPUT:

- `N` – integer

OUTPUT: string

EXAMPLES:

```
sage: SloaneEncyclopedia.sequence_name(1) # optional - sloane_database
'Number of groups of order n.'
```

```
>>> from sage.all import *
>>> SloaneEncyclopedia.sequence_name(Integer(1)) # optional - sloane_database
'Number of groups of order n.'
```

unload()

Remove the database from memory.

sage.databases.sloane.**copy_gz_file**(*gz_source*, *bz_destination*)

Decompress a gzipped file and install the bzipped version.

This is used by SloaneEncyclopedia.install_from_gz to install several gzipped OEIS database files.

INPUT:

- `gz_source` – string; the name of the gzipped file
- `bz_destination` – string; the name of the newly compressed file

1.4 FindStat - the search engine for combinatorial statistics and maps

The interface to the FindStat database is

```
sage: findstat()
The Combinatorial Statistic Finder (https://www.findstat.org/)
```

```
>>> from sage.all import *
>>> findstat()
The Combinatorial Statistic Finder (https://www.findstat.org/)
```

We use the following three notions

- A *combinatorial collection* is a set S with interesting combinatorial properties,
- a *combinatorial map* is a combinatorially interesting map $f : S \to S'$ between combinatorial collections, and
- a *combinatorial statistic* is a combinatorially interesting map $s : S \to \mathbf{Z}$.

You can use the FindStat interface to

- identify a combinatorial statistic or map given the values on a few small objects,
- obtain more terms, formulae, references, etc. for a given statistic or map,
- edit statistics and maps and submit new statistics.

The main entry points to the database are

- *`findstat()`* to search for matching statistics,
- *`findmap()`* to search for matching maps.

1.4.1 A guided tour

Retrieving information

The most straightforward application of the FindStat interface is to gather information about a combinatorial statistic. To do this, we supply *`findstat()`* with a list of `(object, value)` pairs. For example:

```
sage: PM = PerfectMatchings
sage: r = findstat([(m, m.number_of_nestings()) for n in range(6) for m in PM(2*n)],␣
↪depth=1); r # optional -- internet
0: St000042oMp00116 (quality [100, 100])
1: St000041 (quality [20, 100])
...
```

```
>>> from sage.all import *
>>> PM = PerfectMatchings
>>> r = findstat([(m, m.number_of_nestings()) for n in range(Integer(6)) for m in␣
↪PM(Integer(2)*n)], depth=Integer(1)); r # optional -- internet
0: St000042oMp00116 (quality [100, 100])
1: St000041 (quality [20, 100])
...
```

The result of this query is a list (presented as a *sage.databases.oeis.FancyTuple*) of matches. Each match consists of a *FindStatCompoundStatistic* $s : S \to \mathbf{Z}$ and an indication of the quality of the match.

The precise meaning of the result is as follows:

> The composition $f_n \circ \ldots \circ f_2 \circ f_1$ applied to the objects sent to FindStat agrees with all `(object, value)` pairs of s in the database. The optional parameter `depth=1` limits the output to $n = 1$.
>
> Suppose that the quality of the match is (q_a, q_d). Then q_a is the percentage of `(object, value)` pairs that are in the database among those which were sent to FindStat, and q_d is the percentage of `(object, value)` pairs with distinct values in the database among those which were sent to FindStat.

Put differently, if `quality` is not too small it is likely that the statistic sent to FindStat equals $s \circ f_n \circ \ldots \circ f_2 \circ f_1$. If q_a is large, but q_b is small, then there were many matches, but while the sought for statistic attains many distinct values, the match found by FindStat covers only `(object, value)` pairs for few values.

In the case at hand, for the match `St000041`, the list of maps is empty. We can retrieve the description of the statistic from the database as follows:

```
sage: print(r[1].statistic().description())                                    #␣
↪optional -- internet
The number of nestings of a perfect matching.


This is the number of pairs of edges $((a,b), (c,d))$ such that $a\le c\le d\le b$. i.
↪e., the edge $(c,d)$ is nested inside $(a,b)$...
```

```
>>> from sage.all import *
>>> print(r[Integer(1)].statistic().description())                              #␣
↪optional -- internet
The number of nestings of a perfect matching.
<BLANKLINE>
<BLANKLINE>
This is the number of pairs of edges $((a,b), (c,d))$ such that $a\le c\le d\le b$. i.
↪e., the edge $(c,d)$ is nested inside $(a,b)$...
```

We can check the references:

```
sage: r[1].statistic().references()                                            #␣
↪optional -- internet
```

(continues on next page)

(continued from previous page)

```
0: [1]  de Médicis, A., Viennot, X. G., Moments des $q$-polynômes de Laguerre et la␣
↪bijection de Foata-Zeilberger [[MathSciNet:1288802]]
1: [2]  Simion, R., Stanton, D., Octabasic Laguerre polynomials and permutation␣
↪statistics [[MathSciNet:1418763]]...
```

```
>>> from sage.all import *
>>> r[Integer(1)].statistic().references()                                    #␣
↪optional -- internet
0: [1]  de Médicis, A., Viennot, X. G., Moments des $q$-polynômes de Laguerre et la␣
↪bijection de Foata-Zeilberger [[MathSciNet:1288802]]
1: [2]  Simion, R., Stanton, D., Octabasic Laguerre polynomials and permutation␣
↪statistics [[MathSciNet:1418763]]...
```

If you prefer, you can look at this information also in your browser:

```
sage: r[1].statistic().browse()                                             #␣
↪optional -- webbrowser
```

```
>>> from sage.all import *
>>> r[Integer(1)].statistic().browse()                                      #␣
↪optional -- webbrowser
```

Another interesting possibility is to look for equidistributed statistics. Instead of submitting a list of `(object, value)` pairs, we pass a list of pairs `(objects, values)`:

```
sage: data = [(PM(2*n), [m.number_of_nestings() for m in PM(2*n)]) for n in range(5)]
sage: findstat(data, depth=0)                                               #␣
↪optional -- internet
0: St000041 (quality [99, 100])
1: St000042 (quality [99, 100])
```

```
>>> from sage.all import *
>>> data = [(PM(Integer(2)*n), [m.number_of_nestings() for m in PM(Integer(2)*n)])␣
↪for n in range(Integer(5))]
>>> findstat(data, depth=Integer(0))                                        #␣
↪optional -- internet
0: St000041 (quality [99, 100])
1: St000042 (quality [99, 100])
```

This results tells us that the database contains another entry that is equidistributed with the number of nestings on perfect matchings of size at most 10, namely the number of crossings. Note that there is a limit on the number of elements FindStat accepts for a query, which is currently 1000. Queries with more than 1000 elements are truncated.

Let us now look at a slightly more complicated example, where the submitted statistic is the composition of a sequence of combinatorial maps and a statistic known to FindStat. We use the occasion to advertise yet another way to pass values to FindStat:

```
sage: r = findstat(Permutations, lambda pi: pi.saliances()[0], depth=2); r  #␣
↪optional -- internet
0: St000740oMp00066 with offset 1 (quality [100, 100])
1: St000740oMp00087 with offset 1 (quality [100, 100])
...
```

```
>>> from sage.all import *
>>> r = findstat(Permutations, lambda pi: pi.saliances()[Integer(0)],␣
↪depth=Integer(2)); r  # optional -- internet
0: St000740oMp00066 with offset 1 (quality [100, 100])
1: St000740oMp00087 with offset 1 (quality [100, 100])
...
```

Note that some of the matches are up to a global offset. For example, we have:

```
sage: r[1].info()                                                          #␣
↪optional -- internet
after adding 1 to every value
and applying
    Mp00087: inverse first fundamental transformation: Permutations -> Permutations
to the objects (see `.compound_map()` for details)

your input matches
    St000740: The last entry of a permutation.

among the values you sent, 100 percent are actually in the database,
among the distinct values you sent, 100 percent are actually in the database
```

```
>>> from sage.all import *
>>> r[Integer(1)].info()                                                   #␣
↪optional -- internet
after adding 1 to every value
and applying
    Mp00087: inverse first fundamental transformation: Permutations -> Permutations
to the objects (see `.compound_map()` for details)
<BLANKLINE>
your input matches
    St000740: The last entry of a permutation.
<BLANKLINE>
among the values you sent, 100 percent are actually in the database,
among the distinct values you sent, 100 percent are actually in the database
```

Let us pick another particular result:

```
sage: s = findstat("St000051oMp00061oMp00069")                             #␣
↪optional -- internet
sage: s.info()                                                             #␣
↪optional -- internet
    Mp00069: complement: Permutations -> Permutations
    Mp00061: to increasing tree: Permutations -> Binary trees
    St000051: The size of the left subtree of a binary tree.
```

```
>>> from sage.all import *
>>> s = findstat("St000051oMp00061oMp00069")                          # optional -
↪- internet
>>> s.info()                                                          # optional -
↪- internet
    Mp00069: complement: Permutations -> Permutations
```

(continues on next page)

(continued from previous page)

```
    Mp00061: to increasing tree: Permutations -> Binary trees
    St000051: The size of the left subtree of a binary tree.
```

To obtain the value of the statistic sent to FindStat on a given object, apply the maps in the list in the given order to this object, and evaluate the statistic on the result. For example, let us check that the result given by FindStat agrees with our statistic on the following permutation:

```
sage: pi = Permutation([3,1,4,5,2]); pi.saliances()[0]
3
```

```
>>> from sage.all import *
>>> pi = Permutation([Integer(3),Integer(1),Integer(4),Integer(5),Integer(2)]); pi.
↪saliances()[Integer(0)]
3
```

We first have to find out, what the maps and the statistic actually do:

```
sage: print(s.statistic().description())                                    #␣
↪optional -- internet
The size of the left subtree of a binary tree.

sage: print(s.statistic().sage_code())                                      #␣
↪optional -- internet
def statistic(T):
    return T[0].node_number()

sage: print("\n\n".join(m.sage_code() for m in s.compound_map()))           #␣
↪optional -- internet
def mapping(sigma):
    return sigma.complement()

def mapping(sigma):
    return sigma.increasing_tree_shape()
```

```
>>> from sage.all import *
>>> print(s.statistic().description())                                  # optional -
↪- internet
The size of the left subtree of a binary tree.

>>> print(s.statistic().sage_code())                                    # optional -
↪- internet
def statistic(T):
    return T[0].node_number()

>>> print("\n\n".join(m.sage_code() for m in s.compound_map()))         # optional -
↪- internet
def mapping(sigma):
    return sigma.complement()
<BLANKLINE>
def mapping(sigma):
    return sigma.increasing_tree_shape()
```

So, the following should coincide with what we sent FindStat:

```
sage: pi.complement().increasing_tree_shape()[0].node_number()
3
```

```
>>> from sage.all import *
>>> pi.complement().increasing_tree_shape()[Integer(0)].node_number()
3
```

Editing and submitting statistics

Of course, often a statistic will not be in the database:

```
sage: s = findstat([(d, randint(1,1000)) for d in DyckWords(4)]); s          #␣
↪optional -- internet
St000000: a new statistic on Dyck paths
```

```
>>> from sage.all import *
>>> s = findstat([(d, randint(Integer(1),Integer(1000))) for d in␣
↪DyckWords(Integer(4))]); s          # optional -- internet
St000000: a new statistic on Dyck paths
```

In this case, and if the statistic might be "interesting", please consider submitting it to the database using *FindStatStatistic.submit()*.

Also, you may notice omissions, typos or even mistakes in the description, the code and the references. In this case, simply replace the value by using *FindStatFunction.set_description()*, *FindStatStatistic.set_code()* or *FindStatFunction.set_references_raw()*, and then *FindStatStatistic.submit()* your changes for review by the FindStat team.

AUTHORS:

- Martin Rubey (2015): initial version
- Martin Rubey (2020): rewrite, adapt to new FindStat API

class sage.databases.findstat.**FindStat**

Bases: UniqueRepresentation, SageObject

The Combinatorial Statistic Finder.

FindStat is a class preserving user information.

browse()

Open the FindStat web page in a browser.

EXAMPLES:

```
sage: findstat().browse()                                                    #␣
↪optional -- webbrowser
```

```
>>> from sage.all import *
>>> findstat().browse()                                                  # optional -
↪- webbrowser
```

login()

Open the FindStat login page in a browser.

EXAMPLES:

```
sage: findstat().login()                                                    #␣
↪optional -- webbrowser
```

```
>>> from sage.all import *
>>> findstat().login()                                                  # optional -
↪- webbrowser
```

set_user(*name=None*, *email=None*)

Set the user for the session.

INPUT:

- `name` – the name of the user
- `email` – an email address of the user

This information is used when submitting a statistic with *FindStatStatistic.submit()*.

EXAMPLES:

```
sage: findstat().set_user(name='Anonymous', email='invalid@org')
```

```
>>> from sage.all import *
>>> findstat().set_user(name='Anonymous', email='invalid@org')
```

Note

It is usually more convenient to login into the FindStat web page using the *login()* method.

user_email()

Return the user name used for submissions.

EXAMPLES:

```
sage: findstat().set_user(name='Anonymous', email='invalid@org')
sage: findstat().user_email()
'invalid@org'
```

```
>>> from sage.all import *
>>> findstat().set_user(name='Anonymous', email='invalid@org')
>>> findstat().user_email()
'invalid@org'
```

user_name()

Return the user name used for submissions.

EXAMPLES:

```
sage: findstat().set_user(name='Anonymous', email='invalid@org')
sage: findstat().user_name()
'Anonymous'
```

```
>>> from sage.all import *
>>> findstat().set_user(name='Anonymous', email='invalid@org')
>>> findstat().user_name()
'Anonymous'
```

class sage.databases.findstat.**FindStatCollection**(*parent*, *id*, *data*, *sageconstructor_overridden*)

Bases: Element

A FindStat collection.

FindStatCollection is a class representing a combinatorial collection available in the FindStat database.

Its main use is to allow easy specification of the combinatorial collection when using *findstat*. It also provides methods to quickly access its FindStat web page (*browse()*), check whether a particular element is actually in the range considered by FindStat (*in_range()*), etc.

INPUT:

One of the following:

- a string eg. 'Dyck paths' or 'DyckPaths', case-insensitive, or
- an integer designating the FindStat id of the collection, or
- a Sage object belonging to a collection, or
- an iterable producing a Sage object belonging to a collection.

EXAMPLES:

```
sage: from sage.databases.findstat import FindStatCollection
sage: FindStatCollection("Dyck paths")                                  #␣
↪optional -- internet
Cc0005: Dyck paths

sage: FindStatCollection(5)                                             #␣
↪optional -- internet
Cc0005: Dyck paths

sage: FindStatCollection(DyckWord([1,0,1,0]))                           #␣
↪optional -- internet
Cc0005: Dyck paths

sage: FindStatCollection(DyckWords(2))                                  #␣
↪optional -- internet
a subset of Cc0005: Dyck paths

sage: FindStatCollection(DyckWords)                                     #␣
↪optional -- internet
Cc0005: Dyck paths
```

```
>>> from sage.all import *
>>> from sage.databases.findstat import FindStatCollection
>>> FindStatCollection("Dyck paths")                                # optional -
↪- internet
Cc0005: Dyck paths
```

(continues on next page)

(continued from previous page)

```
>>> FindStatCollection(Integer(5))                                          #␣
↪optional -- internet
Cc0005: Dyck paths

>>> FindStatCollection(DyckWord([Integer(1),Integer(0),Integer(1),Integer(0)]))  ␣
↪                        # optional -- internet
Cc0005: Dyck paths

>>> FindStatCollection(DyckWords(Integer(2)))                               #␣
↪optional -- internet
a subset of Cc0005: Dyck paths

>>> FindStatCollection(DyckWords)                                  # optional -
↪- internet
Cc0005: Dyck paths
```

See also

FindStatCollections

browse()

Open the FindStat web page of the collection in a browser.

EXAMPLES:

```
sage: from sage.databases.findstat import FindStatCollection
sage: FindStatCollection("Permutations").browse()                        #␣
↪optional -- webbrowser
```

```
>>> from sage.all import *
>>> from sage.databases.findstat import FindStatCollection
>>> FindStatCollection("Permutations").browse()                  # optional -
↪- webbrowser
```

element_level(*element*)

Return the level of an element.

EXAMPLES:

```
sage: from sage.databases.findstat import FindStatCollection
sage: cc = FindStatCollection("Perfect Matchings")                       #␣
↪optional -- internet
sage: cc.element_level(PerfectMatching([[1,2],[3,4],[5,6]]))             #␣
↪optional -- internet
6
```

```
>>> from sage.all import *
>>> from sage.databases.findstat import FindStatCollection
>>> cc = FindStatCollection("Perfect Matchings")                 # optional -
↪- internet
>>> cc.element_level(PerfectMatching([[Integer(1),Integer(2)],[Integer(3),
```

(continues on next page)

(continued from previous page)

```
↪Integer(4)],[Integer(5),Integer(6)]]))        # optional -- internet
6
```

elements_on_level(*level*)

Return an iterable over the elements on the given level.

EXAMPLES:

```
sage: from sage.databases.findstat import FindStatCollection
sage: FindStatCollection("Perfect Matchings").elements_on_level(4)  #␣
↪optional -- internet
Perfect matchings of {1, 2, 3, 4}
```

```
>>> from sage.all import *
>>> from sage.databases.findstat import FindStatCollection
>>> FindStatCollection("Perfect Matchings").elements_on_level(Integer(4))  #␣
↪optional -- internet
Perfect matchings of {1, 2, 3, 4}
```

first_terms(*function*, *level=None*)

Compute the first few terms of the given function, possibly restricted to a level, as a lazy list.

INPUT:

- `function` – a callable
- `level` – (optional), the level to restrict to

OUTPUT:

A lazy list of pairs of the form `(object, value)`.

EXAMPLES:

```
sage: from sage.databases.findstat import FindStatCollection
sage: c = FindStatCollection("GelfandTsetlinPatterns")                #␣
↪optional -- internet
sage: c.first_terms(lambda x: 1)[:10].list()                          #␣
↪optional -- internet
[([[1, 0], [0]], 1),
 ([[1, 0], [1]], 1),
 ([[2, 0], [0]], 1),
 ([[2, 0], [1]], 1),
 ([[2, 0], [2]], 1),
 ([[1, 1], [1]], 1),
 ([[1, 0, 0], [0, 0], [0]], 1),
 ([[1, 0, 0], [1, 0], [0]], 1),
 ([[1, 0, 0], [1, 0], [1]], 1),
 ([[3, 0], [0]], 1)]
```

```
>>> from sage.all import *
>>> from sage.databases.findstat import FindStatCollection
>>> c = FindStatCollection("GelfandTsetlinPatterns")                # optional -
↪- internet
>>> c.first_terms(lambda x: Integer(1))[:Integer(10)].list()                 ␣
```

(continues on next page)

(continued from previous page)

```
↪      # optional -- internet
[([[1, 0], [0]], 1),
 ([[1, 0], [1]], 1),
 ([[2, 0], [0]], 1),
 ([[2, 0], [1]], 1),
 ([[2, 0], [2]], 1),
 ([[1, 1], [1]], 1),
 ([[1, 0, 0], [0, 0], [0]], 1),
 ([[1, 0, 0], [1, 0], [0]], 1),
 ([[1, 0, 0], [1, 0], [1]], 1),
 ([[3, 0], [0]], 1)]
```

from_string()

Return a function that returns the object given a FindStat representation.

OUTPUT:

The function that produces the Sage object given its FindStat representation as a string.

EXAMPLES:

```
sage: from sage.databases.findstat import FindStatCollection
sage: c = FindStatCollection("Posets")                                  #␣
↪optional -- internet
sage: p = c.from_string()('([(0, 2), (2, 1)], 3)')                      #␣
↪optional -- internet
sage: p.cover_relations()                                               #␣
↪optional -- internet
[[0, 2], [2, 1]]

sage: c = FindStatCollection("Binary Words")                            #␣
↪optional -- internet
sage: w = c.from_string()('010101')                                     #␣
↪optional -- internet
sage: w in c._data["Code"].elements_on_level(6)                         #␣
↪optional -- internet
True
```

```
>>> from sage.all import *
>>> from sage.databases.findstat import FindStatCollection
>>> c = FindStatCollection("Posets")                                # optional -
↪- internet
>>> p = c.from_string()('([(0, 2), (2, 1)], 3)')                    # optional -
↪- internet
>>> p.cover_relations()                                             # optional -
↪- internet
[[0, 2], [2, 1]]

>>> c = FindStatCollection("Binary Words")                          # optional -
↪- internet
>>> w = c.from_string()('010101')                                   # optional -
↪- internet
>>> w in c._data["Code"].elements_on_level(Integer(6))                    #␣
```

(continues on next page)

(continued from previous page)

```
↪optional -- internet
True
```

id()

Return the FindStat identifier of the collection.

OUTPUT: the FindStat identifier of the collection as an integer

EXAMPLES:

```
sage: from sage.databases.findstat import FindStatCollection
sage: c = FindStatCollection("GelfandTsetlinPatterns")                #␣
↪optional -- internet
sage: c.id()                                                          #␣
↪optional -- internet
18
```

```
>>> from sage.all import *
>>> from sage.databases.findstat import FindStatCollection
>>> c = FindStatCollection("GelfandTsetlinPatterns")                # optional -
↪- internet
>>> c.id()                                                          # optional -
↪- internet
18
```

id_str()

Return the FindStat identifier of the collection.

OUTPUT: the FindStat identifier of the collection as a string

EXAMPLES:

```
sage: from sage.databases.findstat import FindStatCollection
sage: c = FindStatCollection("GelfandTsetlinPatterns")                #␣
↪optional -- internet
sage: c.id_str()                                                      #␣
↪optional -- internet
'Cc0018'
```

```
>>> from sage.all import *
>>> from sage.databases.findstat import FindStatCollection
>>> c = FindStatCollection("GelfandTsetlinPatterns")                # optional -
↪- internet
>>> c.id_str()                                                      # optional -
↪- internet
'Cc0018'
```

in_range(*element*)

Check whether an element of the collection is in FindStat's precomputed range.

INPUT:

- `element` – a Sage object that belongs to the collection

OUTPUT: `True`, if `element` is used by the FindStat search engine, and `False` if it is ignored

EXAMPLES:

```
sage: # optional - internet
sage: from sage.databases.findstat import FindStatCollection
sage: c = FindStatCollection("GelfandTsetlinPatterns")
sage: c.in_range(GelfandTsetlinPattern([[2, 1], [1]]))
True
sage: c.in_range(GelfandTsetlinPattern([[3, 1], [1]]))
True
sage: c.in_range(GelfandTsetlinPattern([[7, 1], [1]]))
False
```

```
>>> from sage.all import *
>>> # optional - internet
>>> from sage.databases.findstat import FindStatCollection
>>> c = FindStatCollection("GelfandTsetlinPatterns")
>>> c.in_range(GelfandTsetlinPattern([[Integer(2), Integer(1)],␣
↪[Integer(1)]]))
True
>>> c.in_range(GelfandTsetlinPattern([[Integer(3), Integer(1)],␣
↪[Integer(1)]]))
True
>>> c.in_range(GelfandTsetlinPattern([[Integer(7), Integer(1)],␣
↪[Integer(1)]]))
False
```

is_element(*element*)

Return whether the element belongs to the collection.

If the collection is not yet supported, return whether element is a string.

EXAMPLES:

```
sage: from sage.databases.findstat import FindStatCollection
sage: cc = FindStatCollection("Perfect Matchings")                    #␣
↪optional -- internet
sage: cc.is_element(PerfectMatching([[1,2],[3,4],[5,6]]))             #␣
↪optional -- internet
True

sage: cc.is_element(SetPartition([[1,2],[3,4],[5,6]]))                #␣
↪optional -- internet
False
```

```
>>> from sage.all import *
>>> from sage.databases.findstat import FindStatCollection
>>> cc = FindStatCollection("Perfect Matchings")                    # optional -
↪- internet
>>> cc.is_element(PerfectMatching([[Integer(1),Integer(2)],[Integer(3),
↪Integer(4)],[Integer(5),Integer(6)]]))           # optional -- internet
True

>>> cc.is_element(SetPartition([[Integer(1),Integer(2)],[Integer(3),
↪Integer(4)],[Integer(5),Integer(6)]]))              # optional -- internet
```

(continues on next page)

(continued from previous page)

```
False
```

is_supported()

Check whether the collection is fully supported by the interface.

EXAMPLES:

```
sage: from sage.databases.findstat import FindStatCollection
sage: FindStatCollection(1).is_supported()                                  #␣
↪optional -- internet
True

sage: FindStatCollection(24).is_supported()                                 #␣
↪optional -- internet, random
False
```

```
>>> from sage.all import *
>>> from sage.databases.findstat import FindStatCollection
>>> FindStatCollection(Integer(1)).is_supported()                           #␣
↪optional -- internet
True

>>> FindStatCollection(Integer(24)).is_supported()                          #␣
↪optional -- internet, random
False
```

levels_with_sizes()

Return a dictionary from levels to level sizes.

EXAMPLES:

```
sage: from sage.databases.findstat import FindStatCollection
sage: cc = FindStatCollection("Perfect Matchings")                          #␣
↪optional -- internet
sage: cc.levels_with_sizes()                                                #␣
↪optional -- internet
{2: 1, 4: 3, 6: 15, 8: 105, 10: 945}
```

```
>>> from sage.all import *
>>> from sage.databases.findstat import FindStatCollection
>>> cc = FindStatCollection("Perfect Matchings")                    # optional -
↪- internet
>>> cc.levels_with_sizes()                                          # optional -
↪- internet
{2: 1, 4: 3, 6: 15, 8: 105, 10: 945}
```

name(*style='singular'*)

Return the name of the FindStat collection.

INPUT:

- `style` – string (default: `'singular'`); can be `'singular'`, or `'plural'`

OUTPUT: the name of the FindStat collection, in singular or in plural

EXAMPLES:

```
sage: from sage.databases.findstat import FindStatCollection
sage: FindStatCollection("Binary trees").name()                          #␣
↪optional -- internet
'Binary tree'

sage: FindStatCollection("Binary trees").name(style='plural')            #␣
↪optional -- internet
'Binary trees'
```

```
>>> from sage.all import *
>>> from sage.databases.findstat import FindStatCollection
>>> FindStatCollection("Binary trees").name()                          # optional -
↪- internet
'Binary tree'

>>> FindStatCollection("Binary trees").name(style='plural')            # optional -
↪- internet
'Binary trees'
```

to_string()

Return a function that returns a FindStat representation given an object.

If the collection is not yet supported, return the identity.

OUTPUT:

The function that produces the string representation as needed by the FindStat search webpage.

EXAMPLES:

```
sage: from sage.databases.findstat import FindStatCollection
sage: p = Poset((range(3), [[0, 1], [1, 2]]))                            #␣
↪optional -- internet
sage: c = FindStatCollection("Posets")                                   #␣
↪optional -- internet
sage: c.to_string()(p)                                                   #␣
↪optional -- internet
'([(0, 1), (1, 2)], 3)'
```

```
>>> from sage.all import *
>>> from sage.databases.findstat import FindStatCollection
>>> p = Poset((range(Integer(3)), [[Integer(0), Integer(1)], [Integer(1),␣
↪Integer(2)]]))                          # optional -- internet
>>> c = FindStatCollection("Posets")                                     # optional -
↪- internet
>>> c.to_string()(p)                                                     # optional -
↪- internet
'([(0, 1), (1, 2)], 3)'
```

class sage.databases.findstat.**FindStatCollections**

Bases: UniqueRepresentation, Parent

The class of FindStat collections.

The elements of this class are combinatorial collections in FindStat as of January 2020. If a new collection was added to the web service since then, the dictionary `_SupportedFindStatCollections` in this module has to be updated accordingly.

EXAMPLES:

```
sage: from sage.databases.findstat import FindStatCollections
sage: sorted(c for c in FindStatCollections() if c.is_supported())      #␣
↪optional -- internet
[Cc0001: Permutations,
 Cc0002: Integer partitions,
 Cc0005: Dyck paths,
 Cc0006: Integer compositions,
 Cc0007: Standard tableaux,
 Cc0009: Set partitions,
 Cc0010: Binary trees,
 Cc0012: Perfect matchings,
 Cc0013: Cores,
 Cc0014: Posets,
 Cc0017: Alternating sign matrices,
 Cc0018: Gelfand-Tsetlin patterns,
 Cc0019: Semistandard tableaux,
 Cc0020: Graphs,
 Cc0021: Ordered trees,
 Cc0022: Finite Cartan types,
 Cc0023: Parking functions,
 Cc0024: Binary words,
 Cc0025: Plane partitions,
 Cc0026: Decorated permutations,
 Cc0027: Signed permutations,
 Cc0028: Skew partitions,
 Cc0029: Lattices,
 Cc0030: Ordered set partitions]
```

```
>>> from sage.all import *
>>> from sage.databases.findstat import FindStatCollections
>>> sorted(c for c in FindStatCollections() if c.is_supported())      # optional -
↪- internet
[Cc0001: Permutations,
 Cc0002: Integer partitions,
 Cc0005: Dyck paths,
 Cc0006: Integer compositions,
 Cc0007: Standard tableaux,
 Cc0009: Set partitions,
 Cc0010: Binary trees,
 Cc0012: Perfect matchings,
 Cc0013: Cores,
 Cc0014: Posets,
 Cc0017: Alternating sign matrices,
 Cc0018: Gelfand-Tsetlin patterns,
 Cc0019: Semistandard tableaux,
 Cc0020: Graphs,
 Cc0021: Ordered trees,
 Cc0022: Finite Cartan types,
```

(continues on next page)

(continued from previous page)

```
Cc0023: Parking functions,
Cc0024: Binary words,
Cc0025: Plane partitions,
Cc0026: Decorated permutations,
Cc0027: Signed permutations,
Cc0028: Skew partitions,
Cc0029: Lattices,
Cc0030: Ordered set partitions]
```

Element

alias of *FindStatCollection*

class sage.databases.findstat.**FindStatCombinatorialMap**

Bases: SageObject

A class serving as common ancestor of *FindStatStatistic* and *FindStatCompoundStatistic*.

class sage.databases.findstat.**FindStatCombinatorialStatistic**

Bases: SageObject

A class providing methods to retrieve the first terms of a statistic.

This class provides methods applicable to instances of *FindStatStatistic*, *FindStatCompoundStatistic* and *FindStatStatisticQuery*.

first_terms()

Return the first terms of the (compound) statistic as a dictionary.

OUTPUT:

A dictionary from Sage objects representing an element of the appropriate collection to integers.

This method is overridden in *FindStatStatisticQuery*.

EXAMPLES:

```
sage: findstat(41).first_terms()[PerfectMatching([(1,6),(2,5),(3,4)])]
↪# optional -- internet
3
```

```
>>> from sage.all import *
>>> findstat(Integer(41)).first_terms()[PerfectMatching([(Integer(1),
↪Integer(6)),(Integer(2),Integer(5)),(Integer(3),Integer(4))])]        #␣
↪optional -- internet
3
```

first_terms_str(*max_values=1200*)

Return the first terms of the statistic in the format needed for a FindStat query.

OUTPUT:

A string, where each line is of the form `object => value`, where `object` is the string representation of an element of the appropriate collection as used by FindStat and value is an integer.

EXAMPLES:

```
sage: print(findstat(41).first_terms_str(max_values=4))                    #␣
↪optional -- internet
[(1,2)] => 0
[(1,2),(3,4)] => 0
[(1,3),(2,4)] => 0
[(1,4),(2,3)] => 1
```

```
>>> from sage.all import *
>>> print(findstat(Integer(41)).first_terms_str(max_values=Integer(4)))      ␣
↪      # optional -- internet
[(1,2)] => 0
[(1,2),(3,4)] => 0
[(1,3),(2,4)] => 0
[(1,4),(2,3)] => 1
```

generating_functions(*style='polynomial'*, *max_values=1200*)

Return the generating functions of the statistic as a dictionary.

The keys of this dictionary are the levels for which the generating function of the statistic can be computed from the known data. Each value represents a generating function for one level, as a polynomial, as a dictionary, or as a list of coefficients.

INPUT:

- `style` – string (default: `'polynomial'`); can be `'polynomial'`, `'dictionary'`, or `'list'`

OUTPUT:

- if `style` is `'polynomial'`, the generating function is returned as a polynomial
- if `style` is `'dictionary'`, the generating function is returned as a dictionary representing the monomials of the generating function
- if `style` is `'list'`, the generating function is returned as a list of coefficients of the generating function. In this case, leading and trailing zeros are omitted.

EXAMPLES:

```
sage: st = findstat(41)                                                    #␣
↪optional -- internet
sage: st.generating_functions()                                            #␣
↪optional -- internet
{2: 1,
 4: q + 2,
 6: q^3 + 3*q^2 + 6*q + 5,
 8: q^6 + 4*q^5 + 10*q^4 + 20*q^3 + 28*q^2 + 28*q + 14}

sage: st.generating_functions(style='dictionary')                          #␣
↪optional -- internet
{2: {0: 1},
 4: {0: 2, 1: 1},
 6: {0: 5, 1: 6, 2: 3, 3: 1},
 8: {0: 14, 1: 28, 2: 28, 3: 20, 4: 10, 5: 4, 6: 1}}

sage: st.generating_functions(style='list')                                #␣
↪optional -- internet
{2: [1], 4: [2, 1], 6: [5, 6, 3, 1], 8: [14, 28, 28, 20, 10, 4, 1]}
```

```
>>> from sage.all import *
>>> st = findstat(Integer(41))                                              #␣
↪optional -- internet
>>> st.generating_functions()                                          # optional -
↪- internet
{2: 1,
 4: q + 2,
 6: q^3 + 3*q^2 + 6*q + 5,
 8: q^6 + 4*q^5 + 10*q^4 + 20*q^3 + 28*q^2 + 28*q + 14}

>>> st.generating_functions(style='dictionary')                         # optional -
↪- internet
{2: {0: 1},
 4: {0: 2, 1: 1},
 6: {0: 5, 1: 6, 2: 3, 3: 1},
 8: {0: 14, 1: 28, 2: 28, 3: 20, 4: 10, 5: 4, 6: 1}}

>>> st.generating_functions(style='list')                               # optional -
↪- internet
{2: [1], 4: [2, 1], 6: [5, 6, 3, 1], 8: [14, 28, 28, 20, 10, 4, 1]}
```

oeis_search(*search_size=32*, *verbose=True*)

Search the OEIS for the generating function of the statistic.

INPUT:

- `search_size` – (default: 32) the number of integers in the sequence. If this is chosen too big, the OEIS result may be corrupted.
- `verbose` – boolean (default: `True`); if `True`, some information about the search are printed

OUTPUT: a tuple of OEIS sequences, see *sage.databases.oeis.OEIS.find_by_description()* for more information

EXAMPLES:

```
sage: st = findstat(41)                                                   #␣
↪optional -- internet

sage: st.oeis_search()                                                    #␣
↪optional -- internet
Searching the OEIS for "1  2,1  5,6,3,1  14,28,28,20,10,4,1"
0: A067311: Triangle read by rows: T(n,k) gives number of ways of arranging n␣
↪chords on a circle with k simple intersections ...
```

```
>>> from sage.all import *
>>> st = findstat(Integer(41))                                              #␣
↪optional -- internet

>>> st.oeis_search()                                                   # optional -
↪- internet
Searching the OEIS for "1  2,1  5,6,3,1  14,28,28,20,10,4,1"
0: A067311: Triangle read by rows: T(n,k) gives number of ways of arranging n␣
↪chords on a circle with k simple intersections ...
```

class sage.databases.findstat.**FindStatCompoundMap**(*id*, *domain=None*, *codomain=None*, *check=True*)

Bases: Element, *FindStatCombinatorialMap*

Initialize a compound statistic.

INPUT:

- id – a padded identifier
- domain – (optional), the domain of the compound map
- codomain – (optional), the codomain of the compound map
- check – whether to check that domains and codomains fit

If domain and codomain are given and check is False, they are not fetched from FindStat.

If id is the empty string, domain must be provided, and the identity map on this collection is returned.

browse()

Open the FindStat web page of the compound map in a browser.

EXAMPLES:

```
sage: findmap(62).browse()                                                  #␣
↪optional -- webbrowser
```

```
>>> from sage.all import *
>>> findmap(Integer(62)).browse()                                           #␣
↪optional -- webbrowser
```

codomain()

Return the codomain of the compound map.

EXAMPLES:

```
sage: findmap("Mp00099oMp00127").codomain()                                 #␣
↪optional -- internet
Cc0005: Dyck paths
```

```
>>> from sage.all import *
>>> findmap("Mp00099oMp00127").codomain()                                   # optional -
↪- internet
Cc0005: Dyck paths
```

domain()

Return the domain of the compound map.

EXAMPLES:

```
sage: findmap("Mp00099oMp00127").domain()                                   #␣
↪optional -- internet
Cc0001: Permutations
```

```
>>> from sage.all import *
>>> findmap("Mp00099oMp00127").domain()                                     # optional -
↪- internet
Cc0001: Permutations
```

id_str()

Return the padded identifier of the compound map.

EXAMPLES:

```
sage: findmap("Mp00099oMp00127").id_str()                                   #␣
↪optional -- internet
'Mp00099oMp00127'
```

```
>>> from sage.all import *
>>> findmap("Mp00099oMp00127").id_str()                                  # optional -
↪- internet
'Mp00099oMp00127'
```

info()

Print a detailed explanation of the compound map.

EXAMPLES:

```
sage: findmap("Mp00099oMp00127").info()                                     #␣
↪optional -- internet
    Mp00127: left-to-right-maxima to Dyck path: Permutations -> Dyck paths
    Mp00099: bounce path: Dyck paths -> Dyck paths
```

```
>>> from sage.all import *
>>> findmap("Mp00099oMp00127").info()                                    # optional -
↪- internet
    Mp00127: left-to-right-maxima to Dyck path: Permutations -> Dyck paths
    Mp00099: bounce path: Dyck paths -> Dyck paths
```

maps()

Return the maps occurring in the compound map as a list.

EXAMPLES:

```
sage: findmap("Mp00099oMp00127").maps()                                     #␣
↪optional -- internet
[Mp00127: left-to-right-maxima to Dyck path, Mp00099: bounce path]
```

```
>>> from sage.all import *
>>> findmap("Mp00099oMp00127").maps()                                    # optional -
↪- internet
[Mp00127: left-to-right-maxima to Dyck path, Mp00099: bounce path]
```

class sage.databases.findstat.**FindStatCompoundStatistic**(*id*, *domain=None*, *check=True*)

Bases: Element, *FindStatCombinatorialStatistic*

Initialize a compound statistic.

A compound statistic is a sequence of maps followed by a statistic.

INPUT:

- id – a padded identifier
- domain – (optional), the domain of the compound statistic

- `check` – whether to check that domains and codomains fit

If the domain is given and `check` is `False`, it is not fetched from FindStat.

browse()

Open the FindStat web page of the compound statistic in a browser.

EXAMPLES:

```
sage: from sage.databases.findstat import FindStatCompoundStatistic
sage: FindStatCompoundStatistic("St000042oMp00116").browse()         #␣
↪optional -- webbrowser
```

```
>>> from sage.all import *
>>> from sage.databases.findstat import FindStatCompoundStatistic
>>> FindStatCompoundStatistic("St000042oMp00116").browse()         # optional -
↪- webbrowser
```

compound_map()

Return the compound map which is part of the compound statistic.

EXAMPLES:

```
sage: findstat("St000051oMp00061oMp00069").compound_map()             #␣
↪optional -- internet
Mp00061oMp00069
```

```
>>> from sage.all import *
>>> findstat("St000051oMp00061oMp00069").compound_map()             # optional -
↪- internet
Mp00061oMp00069
```

domain()

Return the domain of the compound statistic.

EXAMPLES:

```
sage: findstat("St000042oMp00116").domain()                           #␣
↪optional -- internet
Cc0012: Perfect matchings
```

```
>>> from sage.all import *
>>> findstat("St000042oMp00116").domain()                           # optional -
↪- internet
Cc0012: Perfect matchings
```

id_str()

Return the padded identifier of the compound statistic.

EXAMPLES:

```
sage: findstat("St000042oMp00116").id_str()                           #␣
↪optional -- internet
'St000042oMp00116'
```

```
>>> from sage.all import *
>>> findstat("St000042oMp00116").id_str()                                    # optional -
↪- internet
'St000042oMp00116'
```

info()

Print a detailed description of the compound statistic.

EXAMPLES:

```
sage: findstat("St000042oMp00116").info()                                    #␣
↪optional -- internet
    Mp00116: Kasraoui-Zeng: Perfect matchings -> Perfect matchings
    St000042: The number of crossings of a perfect matching.
```

```
>>> from sage.all import *
>>> findstat("St000042oMp00116").info()                                      # optional -
↪- internet
    Mp00116: Kasraoui-Zeng: Perfect matchings -> Perfect matchings
    St000042: The number of crossings of a perfect matching.
```

statistic()

Return the statistic of the compound statistic.

EXAMPLES:

```
sage: findstat("St000041oMp00116").statistic()                               #␣
↪optional -- internet
St000041: The number of nestings of a perfect matching.
```

```
>>> from sage.all import *
>>> findstat("St000041oMp00116").statistic()                                 # optional -
↪- internet
St000041: The number of nestings of a perfect matching.
```

class sage.databases.findstat.**FindStatFunction**(*id*, *data=None*, *function=None*)

Bases: SageObject

A class providing the common methods of *FindStatMap* and *FindStatStatistic*.

This class provides methods to access and modify properties of a single statistic or map of the FindStat database.

description()

Return the description of the statistic or map.

OUTPUT: string; for statistics, the first line is used as name

EXAMPLES:

```
sage: print(findstat(51).description())                                      #␣
↪optional -- internet
The size of the left subtree of a binary tree.
```

```
>>> from sage.all import *
>>> print(findstat(Integer(51)).description())                                #␣
```

(continues on next page)

(continued from previous page)

```
↪optional -- internet
The size of the left subtree of a binary tree.
```

domain()

Return the FindStat domain of the statistic or map.

OUTPUT:

The domain of the statistic or map as an instance of *FindStatCollection*.

EXAMPLES:

```
sage: findstat(51).domain()                                                #␣
↪optional -- internet
Cc0010: Binary trees

sage: findmap(62).domain()                                                 #␣
↪optional -- internet
Cc0001: Permutations
```

```
>>> from sage.all import *
>>> findstat(Integer(51)).domain()                                         #␣
↪optional -- internet
Cc0010: Binary trees

>>> findmap(Integer(62)).domain()                                          #␣
↪optional -- internet
Cc0001: Permutations
```

id()

Return the FindStat identifier of the statistic or map.

OUTPUT: the FindStat identifier of the statistic or map, as an integer

EXAMPLES:

```
sage: findstat(51).id()                                                    #␣
↪optional -- internet
51
```

```
>>> from sage.all import *
>>> findstat(Integer(51)).id()                                             #␣
↪optional -- internet
51
```

id_str()

Return the FindStat identifier of the statistic or map.

OUTPUT: the FindStat identifier of the statistic or map, as a string

EXAMPLES:

```
sage: findstat(51).id_str()                                                #␣
↪optional -- internet
'St000051'
```

```
>>> from sage.all import *
>>> findstat(Integer(51)).id_str()                                          #␣
↪optional -- internet
'St000051'
```

name()

Return the name of the statistic or map.

OUTPUT:

A string. For statistics, this is just the first line of the description.

EXAMPLES:

```
sage: findstat(51).name()                                                 #␣
↪optional -- internet
'The size of the left subtree of a binary tree.'
```

```
>>> from sage.all import *
>>> findstat(Integer(51)).name()                                          #␣
↪optional -- internet
'The size of the left subtree of a binary tree.'
```

references()

Return the references associated with the statistic or map.

OUTPUT:

An instance of *sage.databases.oeis.FancyTuple*, each item corresponds to a reference.

EXAMPLES:

```
sage: findstat(41).references()                                           #␣
↪optional -- internet
0: [1]  de Médicis, A., Viennot, X. G., Moments des $q$-polynômes de Laguerre␣
↪et la bijection de Foata-Zeilberger [[MathSciNet:1288802]]
1: [2]  Simion, R., Stanton, D., Octabasic Laguerre polynomials and␣
↪permutation statistics [[MathSciNet:1418763]]
```

```
>>> from sage.all import *
>>> findstat(Integer(41)).references()                                    #␣
↪optional -- internet
0: [1]  de Médicis, A., Viennot, X. G., Moments des $q$-polynômes de Laguerre␣
↪et la bijection de Foata-Zeilberger [[MathSciNet:1288802]]
1: [2]  Simion, R., Stanton, D., Octabasic Laguerre polynomials and␣
↪permutation statistics [[MathSciNet:1418763]]
```

references_raw()

Return the unrendered references associated with the statistic or map.

EXAMPLES:

```
sage: print(findstat(41).references_raw())                                #␣
↪optional -- internet
[1]  [[MathSciNet:1288802]]
[2]  [[MathSciNet:1418763]]
```

```
>>> from sage.all import *
>>> print(findstat(Integer(41)).references_raw())                          #␣
↪optional -- internet
[1]  [[MathSciNet:1288802]]
[2]  [[MathSciNet:1418763]]
```

reset()

Discard all modification of the statistic or map.

EXAMPLES:

```
sage: s = findmap(62)                                                    #␣
↪optional -- internet
sage: s.set_name(u"Möbius"); s                                           #␣
↪optional -- internet
Mp00062(modified): Möbius
sage: s.reset(); s                                                       #␣
↪optional -- internet
Mp00062: Lehmer-code to major-code bijection
```

```
>>> from sage.all import *
>>> s = findmap(Integer(62))                                             #␣
↪optional -- internet
>>> s.set_name(u"Möbius"); s                                        # optional -
↪- internet
Mp00062(modified): Möbius
>>> s.reset(); s                                                    # optional -
↪- internet
Mp00062: Lehmer-code to major-code bijection
```

sage_code()

Return the Sage code associated with the statistic or map.

OUTPUT: an empty string or a string of the form:

```
def statistic(x):
    ...
```

or:

```
def mapping(x):
    ...
```

EXAMPLES:

```
sage: print(findstat(51).sage_code())                                    #␣
↪optional -- internet
def statistic(T):
    return T[0].node_number()
```

```
>>> from sage.all import *
>>> print(findstat(Integer(51)).sage_code())                             #␣
↪optional -- internet
```

(continues on next page)

(continued from previous page)

```
def statistic(T):
    return T[0].node_number()
```

set_description(*value*)

Set the description of the statistic or map.

INPUT:

- `value` – string; for statistics, this is the name of the statistic followed by its description on a separate line

This information is used when submitting the statistic or map with `submit()`.

EXAMPLES:

```
sage: q = findstat([(d, randint(1, 1000)) for d in DyckWords(4)])
↪# optional -- internet
sage: q.set_description("Random values on Dyck paths.\nNot for submission.")
↪# optional -- internet
sage: print(q.description())
↪# optional -- internet
Random values on Dyck paths.
Not for submission.
```

```
>>> from sage.all import *
>>> q = findstat([(d, randint(Integer(1), Integer(1000))) for d in␣
↪DyckWords(Integer(4))])              # optional -- internet
>>> q.set_description("Random values on Dyck paths.\nNot for submission.")  #␣
↪optional -- internet
>>> print(q.description())                                                  #␣
↪optional -- internet
Random values on Dyck paths.
Not for submission.
```

set_references_raw(*value*)

Set the references associated with the statistic or map.

INPUT:

- `value` – string; each reference should be on a single line, and consist of one or more links to the same item

FindStat will automatically resolve the links, if possible. A complete list of supported services can be found at <https://findstat.org/NewStatistic>.

This information is used when submitting the statistic with `submit()`.

EXAMPLES:

```
sage: q = findstat([(d, randint(1, 1000)) for d in DyckWords(4)])    #␣
↪optional -- internet
sage: q.set_references_raw("[[arXiv:1102.4226]]\n[[oeis:A000001]]") #␣
↪optional -- internet
sage: q.references()                                                 #␣
↪optional -- internet
0: [[arXiv:1102.4226]]
1: [[oeis:A000001]]
```

```
>>> from sage.all import *
>>> q = findstat([(d, randint(Integer(1), Integer(1000))) for d in␣
↪DyckWords(Integer(4))])    # optional -- internet
>>> q.set_references_raw("[[arXiv:1102.4226]]\n[[oeis:A000001]]") # optional -
↪- internet
>>> q.references()                                                  # optional -
↪- internet
0: [[arXiv:1102.4226]]
1: [[oeis:A000001]]
```

set_sage_code(*value*)

Set the code associated with the statistic or map.

INPUT:

- `value` – string; SageMath code producing the values of the statistic or map

Contributors are encouraged to submit code for statistics using *FindStatStatistic.set_code()*. Modifying the "verified" SageMath code using this method is restricted to members of the FindStatCrew, for all other contributors this method has no effect.

EXAMPLES:

```
sage: q = findstat([(d, randint(1,1000)) for d in DyckWords(4)])
↪# optional -- internet
sage: q.set_sage_code("def statistic(x):\n    return randint(1, 1000)")      ␣
↪ # optional -- internet
sage: print(q.sage_code())
↪# optional -- internet
def statistic(x):
    return randint(1,1000)
```

```
>>> from sage.all import *
>>> q = findstat([(d, randint(Integer(1),Integer(1000))) for d in␣
↪DyckWords(Integer(4))])              # optional -- internet
>>> q.set_sage_code("def statistic(x):\n    return randint(1, 1000)")
↪# optional -- internet
>>> print(q.sage_code())                                                  #␣
↪optional -- internet
def statistic(x):
    return randint(1,1000)
```

class sage.databases.findstat.**FindStatMap**(*parent*, *id*)

Bases: `Element`, *FindStatFunction*, *FindStatCombinatorialMap*

A FindStat map.

FindStatMap is a class representing a combinatorial map available in the FindStat database.

This class provides methods to inspect various properties of these maps, in particular `code()`.

EXAMPLES:

```
sage: from sage.databases.findstat import FindStatMap
sage: FindStatMap(116)                                                    #␣
↪optional -- internet
Mp00116: Kasraoui-Zeng
```

```
>>> from sage.all import *
>>> from sage.databases.findstat import FindStatMap
>>> FindStatMap(Integer(116))                                                  #␣
↪optional -- internet
Mp00116: Kasraoui-Zeng
```

See also

FindStatMaps

browse()

Open the FindStat web page of the map in a browser.

EXAMPLES:

```
sage: from sage.databases.findstat import FindStatMap
sage: FindStatMap(116).browse()                                             #␣
↪optional -- webbrowser
```

```
>>> from sage.all import *
>>> from sage.databases.findstat import FindStatMap
>>> FindStatMap(Integer(116)).browse()                                        #␣
↪optional -- webbrowser
```

codomain()

Return the FindStat collection which is the codomain of the map.

OUTPUT: the codomain of the map as a *FindStatCollection*

EXAMPLES:

```
sage: from sage.databases.findstat import FindStatMap                       #␣
↪optional -- internet
sage: FindStatMap(27).codomain()                                            #␣
↪optional -- internet
Cc0002: Integer partitions
```

```
>>> from sage.all import *
>>> from sage.databases.findstat import FindStatMap                 # optional -
↪- internet
>>> FindStatMap(Integer(27)).codomain()                                       #␣
↪optional -- internet
Cc0002: Integer partitions
```

edit()

Open the FindStat web page for editing the map in a browser.

info()

Print a detailed description of the map.

EXAMPLES:

```
sage: findmap("Mp00116").info()                                          #␣
↪optional -- internet
    Mp00116: Kasraoui-Zeng: Perfect matchings -> Perfect matchings
```

```
>>> from sage.all import *
>>> findmap("Mp00116").info()                                          # optional -
↪- internet
    Mp00116: Kasraoui-Zeng: Perfect matchings -> Perfect matchings
```

properties_raw()

Return the properties of the map.

OUTPUT: the properties as a string

EXAMPLES:

```
sage: from sage.databases.findstat import FindStatMap                    #␣
↪optional -- internet
sage: FindStatMap(61).properties_raw()                                   #␣
↪optional -- internet
'surjective, graded'
```

```
>>> from sage.all import *
>>> from sage.databases.findstat import FindStatMap                    # optional -
↪- internet
>>> FindStatMap(Integer(61)).properties_raw()                                #␣
↪optional -- internet
'surjective, graded'
```

set_name(*value*)

Set the name of the map.

INPUT:

- `value` – string; the new name of the map

This information is used when submitting the map with *submit()*.

set_properties_raw(*value*)

Set the properties of the map.

EXAMPLES:

```
sage: # optional - internet
sage: from sage.databases.findstat import FindStatMap
sage: FindStatMap(61).set_properties_raw('surjective')
sage: FindStatMap(61).properties_raw()
'surjective'
sage: FindStatMap(61)
Mp00061(modified): to increasing tree
sage: FindStatMap(61).reset()
sage: FindStatMap(61)
Mp00061: to increasing tree
```

```
>>> from sage.all import *
>>> # optional - internet
>>> from sage.databases.findstat import FindStatMap
>>> FindStatMap(Integer(61)).set_properties_raw('surjective')
>>> FindStatMap(Integer(61)).properties_raw()
'surjective'
>>> FindStatMap(Integer(61))
Mp00061(modified): to increasing tree
>>> FindStatMap(Integer(61)).reset()
>>> FindStatMap(Integer(61))
Mp00061: to increasing tree
```

submit()

Open the FindStat web page for editing the map in a browser.

class sage.databases.findstat.**FindStatMapQuery**(*data=None*, *values_of=None*, *distribution_of=None*, *domain=None*, *codomain=None*, *known_terms=None*, *function=None*, *depth=2*, *debug=False*)

Bases: *FindStatMap*

A class representing a query for FindStat (compound) maps.

class sage.databases.findstat.**FindStatMaps**(*domain=None*, *codomain=None*)

Bases: UniqueRepresentation, Parent

The class of FindStat maps.

The elements of this class are combinatorial maps currently in FindStat.

EXAMPLES:

We can print a sample map for each domain and codomain:

```
sage: from sage.databases.findstat import FindStatCollections, FindStatMaps
sage: ccs = sorted(FindStatCollections())[:3]                                  #␣
↪optional -- internet
sage: for cc_dom in ccs:                                                       #␣
↪optional -- internet
....:     for cc_codom in ccs:
....:         print(cc_dom.name(style='plural') + " -> " + cc_codom.name(style=
↪'plural'))
....:         try:
....:             print("    " + next(iter(FindStatMaps(cc_dom, cc_codom))).
↪name())
....:         except StopIteration:
....:             pass
Permutations -> Permutations
    Lehmer-code to major-code bijection
Permutations -> Integer partitions
    Robinson-Schensted tableau shape
Permutations -> Dyck paths
    left-to-right-maxima to Dyck path
Integer partitions -> Permutations
Integer partitions -> Integer partitions
    conjugate
Integer partitions -> Dyck paths
```

(continues on next page)

(continued from previous page)

```
    to Dyck path
Dyck paths -> Permutations
    to non-crossing permutation
Dyck paths -> Integer partitions
    to partition
Dyck paths -> Dyck paths
    reverse
```

```
>>> from sage.all import *
>>> from sage.databases.findstat import FindStatCollections, FindStatMaps
>>> ccs = sorted(FindStatCollections())[:Integer(3)]                          #␣
↪optional -- internet
>>> for cc_dom in ccs:                                                        # optional -
↪- internet
...     for cc_codom in ccs:
...         print(cc_dom.name(style='plural') + " -> " + cc_codom.name(style=
↪'plural'))
...         try:
...             print("    " + next(iter(FindStatMaps(cc_dom, cc_codom))).name())
...         except StopIteration:
...             pass
Permutations -> Permutations
    Lehmer-code to major-code bijection
Permutations -> Integer partitions
    Robinson-Schensted tableau shape
Permutations -> Dyck paths
    left-to-right-maxima to Dyck path
Integer partitions -> Permutations
Integer partitions -> Integer partitions
    conjugate
Integer partitions -> Dyck paths
    to Dyck path
Dyck paths -> Permutations
    to non-crossing permutation
Dyck paths -> Integer partitions
    to partition
Dyck paths -> Dyck paths
    reverse
```

Element

alias of *FindStatMap*

class sage.databases.findstat.**FindStatMatchingMap**(*matching_map*, *quality*, *domain=None*, *codomain=None*)

Bases: *FindStatCompoundMap*

Initialize a FindStat map match.

INPUT:

- `matching_map` – a compound map identifier
- `quality` – the quality of the match, as provided by FindStat
- `domain` – (optional) the domain of the compound map

- `codomain` – (optional), the codomain of the compound map

EXAMPLES:

```
sage: from sage.databases.findstat import FindStatMatchingMap
sage: FindStatMatchingMap("Mp00099oMp00127", [83])                    # optional --␣
↪internet
Mp00099oMp00127 (quality [83])
```

```
>>> from sage.all import *
>>> from sage.databases.findstat import FindStatMatchingMap
>>> FindStatMatchingMap("Mp00099oMp00127", [Integer(83)])                    #␣
↪optional -- internet
Mp00099oMp00127 (quality [83])
```

info()

Print a detailed explanation of the match.

EXAMPLES:

```
sage: from sage.databases.findstat import FindStatMatchingMap
sage: FindStatMatchingMap("Mp00099oMp00127", [83]).info()             #␣
↪optional -- internet
your input matches
    Mp00127: left-to-right-maxima to Dyck path: Permutations -> Dyck paths
    Mp00099: bounce path: Dyck paths -> Dyck paths

among the values you sent, 83 percent are actually in the database
```

```
>>> from sage.all import *
>>> from sage.databases.findstat import FindStatMatchingMap
>>> FindStatMatchingMap("Mp00099oMp00127", [Integer(83)]).info()             #␣
↪optional -- internet
your input matches
    Mp00127: left-to-right-maxima to Dyck path: Permutations -> Dyck paths
    Mp00099: bounce path: Dyck paths -> Dyck paths
<BLANKLINE>
among the values you sent, 83 percent are actually in the database
```

quality()

Return the quality of the match, as provided by FindStat.

EXAMPLES:

```
sage: from sage.databases.findstat import FindStatMatchingMap
sage: FindStatMatchingMap("Mp00099oMp00127", [83]).quality()          #␣
↪optional -- internet
[83]
```

```
>>> from sage.all import *
>>> from sage.databases.findstat import FindStatMatchingMap
>>> FindStatMatchingMap("Mp00099oMp00127", [Integer(83)]).quality()          #␣
↪optional -- internet
[83]
```

class sage.databases.findstat.**FindStatMatchingStatistic**(*matching_statistic*, *offset*, *quality*, *domain=None*)

Bases: *FindStatCompoundStatistic*

Initialize a FindStat statistic match.

INPUT:

- `matching_statistic` – a compound statistic identifier
- `offset` – the offset of the values, as provided by FindStat
- `quality` – the quality of the match, as provided by FindStat
- `domain` – (optional) the domain of the compound statistic

EXAMPLES:

```
sage: from sage.databases.findstat import FindStatMatchingStatistic
sage: FindStatMatchingStatistic("St000042oMp00116", 1, [17, 83])    # optional --␣
↪internet
St000042oMp00116 with offset 1 (quality [17, 83])
```

```
>>> from sage.all import *
>>> from sage.databases.findstat import FindStatMatchingStatistic
>>> FindStatMatchingStatistic("St000042oMp00116", Integer(1), [Integer(17),␣
↪Integer(83)])    # optional -- internet
St000042oMp00116 with offset 1 (quality [17, 83])
```

info()

Print a detailed explanation of the match.

EXAMPLES:

```
sage: from sage.databases.findstat import FindStatMatchingStatistic
sage: r = FindStatMatchingStatistic("St000042oMp00116", 1, [17, 83])
↪# optional -- internet
sage: r.info()                                                          #␣
↪optional -- internet
after adding 1 to every value
and applying
    Mp00116: Kasraoui-Zeng: Perfect matchings -> Perfect matchings
to the objects (see `.compound_map()` for details)

your input matches
    St000042: The number of crossings of a perfect matching.

among the values you sent, 17 percent are actually in the database,
among the distinct values you sent, 83 percent are actually in the database

sage: r = FindStatMatchingStatistic("St000042", 1, [17, 83])           #␣
↪optional -- internet
sage: r.info()                                                          #␣
↪optional -- internet
after adding 1 to every value

your input matches
```

(continues on next page)

(continued from previous page)

```
    St000042: The number of crossings of a perfect matching.

among the values you sent, 17 percent are actually in the database,
among the distinct values you sent, 83 percent are actually in the database
```

```
>>> from sage.all import *
>>> from sage.databases.findstat import FindStatMatchingStatistic
>>> r = FindStatMatchingStatistic("St000042oMp00116", Integer(1),␣
↪[Integer(17), Integer(83)])          # optional -- internet
>>> r.info()                                                    # optional -
↪- internet
after adding 1 to every value
and applying
    Mp00116: Kasraoui-Zeng: Perfect matchings -> Perfect matchings
to the objects (see `.compound_map()` for details)
<BLANKLINE>
your input matches
    St000042: The number of crossings of a perfect matching.
<BLANKLINE>
among the values you sent, 17 percent are actually in the database,
among the distinct values you sent, 83 percent are actually in the database

>>> r = FindStatMatchingStatistic("St000042", Integer(1), [Integer(17),␣
↪Integer(83)])        # optional -- internet
>>> r.info()                                                    # optional -
↪- internet
after adding 1 to every value
<BLANKLINE>
your input matches
    St000042: The number of crossings of a perfect matching.
<BLANKLINE>
among the values you sent, 17 percent are actually in the database,
among the distinct values you sent, 83 percent are actually in the database
```

offset()

Return the offset which has to be added to each value of the compound statistic to obtain the desired value.

EXAMPLES:

```
sage: from sage.databases.findstat import FindStatMatchingStatistic
sage: r = FindStatMatchingStatistic("St000042oMp00116", 1, [17, 83])
↪# optional -- internet
sage: r.offset()                                                  #␣
↪optional -- internet
1
```

```
>>> from sage.all import *
>>> from sage.databases.findstat import FindStatMatchingStatistic
>>> r = FindStatMatchingStatistic("St000042oMp00116", Integer(1),␣
↪[Integer(17), Integer(83)])          # optional -- internet
>>> r.offset()                                                  # optional -
↪- internet
```

(continues on next page)

(continued from previous page)

```
1
```

quality()

Return the quality of the match, as provided by FindStat.

The quality of a statistic match is a pair of percentages (q_a, q_d), where q_a is the percentage of `(object, value)` pairs that are in the database among those which were sent to FindStat, and q_d is the percentage of `(object, value)` pairs with distinct values in the database among those which were sent to FindStat.

EXAMPLES:

```
sage: from sage.databases.findstat import FindStatMatchingStatistic
sage: r = FindStatMatchingStatistic("St000042oMp00116", 1, [17, 83])
↪# optional -- internet
sage: r.quality()                                                    #␣
↪optional -- internet
[17, 83]
```

```
>>> from sage.all import *
>>> from sage.databases.findstat import FindStatMatchingStatistic
>>> r = FindStatMatchingStatistic("St000042oMp00116", Integer(1),␣
↪[Integer(17), Integer(83)])          # optional -- internet
>>> r.quality()                                                  # optional -
↪- internet
[17, 83]
```

class sage.databases.findstat.**FindStatStatistic**(*parent*, *id*)

Bases: `Element`, *FindStatFunction*, *FindStatCombinatorialStatistic*

A FindStat statistic.

FindStatStatistic is a class representing a combinatorial statistic available in the FindStat database.

This class provides methods to inspect and update various properties of these statistics.

EXAMPLES:

```
sage: from sage.databases.findstat import FindStatStatistic
sage: FindStatStatistic(41)                                          #␣
↪optional -- internet
St000041: The number of nestings of a perfect matching.
```

```
>>> from sage.all import *
>>> from sage.databases.findstat import FindStatStatistic
>>> FindStatStatistic(Integer(41))                                    #␣
↪optional -- internet
St000041: The number of nestings of a perfect matching.
```

See also

FindStatStatistics

browse()

Open the FindStat web page of the statistic in a browser.

EXAMPLES:

```
sage: findstat(41).browse()                                                 #␣
↪optional -- webbrowser
```

```
>>> from sage.all import *
>>> findstat(Integer(41)).browse()                                          #␣
↪optional -- webbrowser
```

code()

Return the code associated with the statistic or map.

OUTPUT: string

Contributors are encouraged to submit Sage code in the form:

```
def statistic(x):
    ...
```

but the string may also contain code for other computer algebra systems.

EXAMPLES:

```
sage: print(findstat(41).code())                                            #␣
↪optional -- internet
def statistic(x):
    return len(x.nestings())

sage: print(findstat(118).code())                                           #␣
↪optional -- internet
(* in Mathematica *)
tree = {{{{}, {}}, {{}, {}}}, {{{}, {}}, {{}, {}}}};
Count[tree, {{___}, {{___}, {{___}, {___}}}}, {0, Infinity}]
```

```
>>> from sage.all import *
>>> print(findstat(Integer(41)).code())                                     #␣
↪optional -- internet
def statistic(x):
    return len(x.nestings())

>>> print(findstat(Integer(118)).code())                                    #␣
↪optional -- internet
(* in Mathematica *)
tree = {{{{}, {}}, {{}, {}}}, {{{}, {}}, {{}, {}}}};
Count[tree, {{___}, {{___}, {{___}, {___}}}}, {0, Infinity}]
```

edit(*max_values=1200*)

Open the FindStat web page for editing the statistic or submitting a new statistic in a browser.

info()

Print a detailed description of the statistic.

EXAMPLES:

```
sage: findstat("St000042").info()                                          #␣
↪optional -- internet
    St000042: The number of crossings of a perfect matching.
```

```
>>> from sage.all import *
>>> findstat("St000042").info()                                        # optional -
↪- internet
    St000042: The number of crossings of a perfect matching.
```

set_code(*value*)

Set the code associated with the statistic.

INPUT:

- `value` – string; code producing the values of the statistic

Contributors are encouraged to submit SageMath code in the form:

```
def statistic(x):
    ...
```

However, code for any other platform is accepted also.

This information is used when submitting the statistic with *submit()*.

EXAMPLES:

```
sage: q = findstat([(d, randint(1,1000)) for d in DyckWords(4)])     #␣
↪optional -- internet
sage: q.set_code("def statistic(x):\n    return randint(1,1000)")    #␣
↪optional -- internet
sage: print(q.code())                                                #␣
↪optional -- internet
def statistic(x):
    return randint(1,1000)
```

```
>>> from sage.all import *
>>> q = findstat([(d, randint(Integer(1),Integer(1000))) for d in␣
↪DyckWords(Integer(4))])     # optional -- internet
>>> q.set_code("def statistic(x):\n    return randint(1,1000)")    # optional -
↪- internet
>>> print(q.code())                                                 # optional -
↪- internet
def statistic(x):
    return randint(1,1000)
```

set_first_terms(*values*)

Update the first terms of the statistic.

INPUT:

- `values` – list of pairs of the form `(object, value)` where `object` is a Sage object representing an element of the appropriate collection and `value` is an integer

This information is used when submitting the statistic with *submit()*.

Warning

This method cannot check whether the given values are actually correct. Moreover, it does not even perform any sanity checks.

submit(*max_values=1200*)

Open the FindStat web page for editing the statistic or submitting a new statistic in a browser.

class sage.databases.findstat.**FindStatStatisticQuery**(*data=None*, *values_of=None*, *distribution_of=None*, *domain=None*, *known_terms=None*, *function=None*, *depth=2*, *debug=False*)

Bases: *FindStatStatistic*

A class representing a query for FindStat (compound) statistics.

first_terms(*max_values=1200*)

Return the pairs of the known terms which contain singletons as a dictionary.

EXAMPLES:

```
sage: PM = PerfectMatchings
sage: l = [(PM(2*n), [m.number_of_nestings() for m in PM(2*n)]) for n in␣
↪range(5)]
sage: r = findstat(l, depth=0); r                                    # optional␣
↪-- internet
0: St000041 (quality [99, 100])
1: St000042 (quality [99, 100])
sage: r.first_terms()                                                # optional␣
↪-- internet
{[]: 0, [(1, 2)]: 0}
```

```
>>> from sage.all import *
>>> PM = PerfectMatchings
>>> l = [(PM(Integer(2)*n), [m.number_of_nestings() for m in␣
↪PM(Integer(2)*n)]) for n in range(Integer(5))]
>>> r = findstat(l, depth=Integer(0)); r                                    #␣
↪optional -- internet
0: St000041 (quality [99, 100])
1: St000042 (quality [99, 100])
>>> r.first_terms()                                                # optional --
↪ internet
{[]: 0, [(1, 2)]: 0}
```

class sage.databases.findstat.**FindStatStatistics**(*domain=None*)

Bases: UniqueRepresentation, Parent

The class of FindStat statistics.

The elements of this class are combinatorial statistics currently in FindStat.

EXAMPLES:

We can print a list of the first few statistics currently in FindStat in a given domain:

```
sage: from sage.databases.findstat import FindStatStatistics
sage: for st, _ in zip(FindStatStatistics("Perfect Matchings"), range(3)):
↪# optional -- internet
....:     print("    " + st.name())
The number of nestings of a perfect matching.
The number of crossings of a perfect matching.
The number of crossings plus two-nestings of a perfect matching.
```

```
>>> from sage.all import *
>>> from sage.databases.findstat import FindStatStatistics
>>> for st, _ in zip(FindStatStatistics("Perfect Matchings"), range(Integer(3))):␣
↪       # optional -- internet
...     print("    " + st.name())
The number of nestings of a perfect matching.
The number of crossings of a perfect matching.
The number of crossings plus two-nestings of a perfect matching.
```

Element

alias of *FindStatStatistic*

sage.databases.findstat.**findmap**(**args*, ***kwargs*)

Return matching maps.

INPUT:

Valid keywords are: `domain`, `codomain`, `values`, `distribution`, `depth` and `max_values`. They have the following meanings:

- `depth` – nonnegative integer (default: `FINDSTAT_DEFAULT_DEPTH`); specifying how many maps to apply to generate the given map
- `max_values` – integer (default: `FINDSTAT_MAX_VALUES`); specifying how many values are sent to the finder
- `domain`, `codomain` – integer or string of the form `Cc1234`, designates the domain and codomain of the sought for maps
- `values`, `distribution` – data specifying the values or distribution of values of the sought for maps. The keyword arguments `depth` and `max_values` are passed to the finder. The data may be specified in one of the following forms:
  - a list of pairs of the form `(object, value)`, or a dictionary from Sage objects to Sage objects.
  - a list of pairs of the form `(list of objects, list of values)`, or a single pair of the form `(list of objects, list of values)`. In each pair there should be as many objects as values.
  - a callable. In this case, the domain must be specified, also. The callable is then used to generate `max_values (object, value)` pairs.

    The number of terms generated may also be controlled by passing an iterable collection, such as `Permutations(3)`.

`findmap` also accepts at most three positional arguments as follows:

- a single positional argument, if none of `domain`, `codomain`, `values` or `distribution` are specified, is interpreted as a FindStat map identifier. If further arguments are given and it is a string, it is interpreted as a domain. If all this fails, it is interpreted as the specification of values.
- if two positional arguments are given, the first is interpreted as domain, and the second either as codomain or the specification of values.

- if three positional arguments are given, the first two are interpreted as domain and codomain, and the third as the specification of values.

OUTPUT:

An instance of a *FindStatMap*, *FindStatMapQuery* or *FindStatMaps*.

EXAMPLES:

A particular map can be retrieved by its Mp-identifier or number:

```
sage: findmap('Mp00062')                                                    #␣
↪optional -- internet
Mp00062: Lehmer-code to major-code bijection

sage: findmap(62)                                                           #␣
↪optional -- internet
Mp00062: Lehmer-code to major-code bijection

sage: findmap("Mp00099oMp00127")                                            #␣
↪optional -- internet
Mp00099oMp00127
```

```
>>> from sage.all import *
>>> findmap('Mp00062')                                                  # optional -
↪- internet
Mp00062: Lehmer-code to major-code bijection

>>> findmap(Integer(62))                                                      #␣
↪optional -- internet
Mp00062: Lehmer-code to major-code bijection

>>> findmap("Mp00099oMp00127")                                          # optional -
↪- internet
Mp00099oMp00127
```

The database can be searched by providing a list of pairs:

```
sage: l = [pi for n in range(5) for pi in Permutations(n)]
sage: findmap([(pi, pi.complement().increasing_tree_shape()) for pi in l],␣
↪depth=2) # optional -- internet
0: Mp00061oMp00069 (quality [...])
```

```
>>> from sage.all import *
>>> l = [pi for n in range(Integer(5)) for pi in Permutations(n)]
>>> findmap([(pi, pi.complement().increasing_tree_shape()) for pi in l],␣
↪depth=Integer(2)) # optional -- internet
0: Mp00061oMp00069 (quality [...])
```

or a dictionary:

```
sage: findmap({pi: pi.complement().increasing_tree_shape() for pi in l}, depth=2)
↪# optional -- internet
0: Mp00061oMp00069 (quality [...])
```

```
>>> from sage.all import *
>>> findmap({pi: pi.complement().increasing_tree_shape() for pi in l},␣
↪depth=Integer(2)) # optional -- internet
0: Mp00061oMp00069 (quality [...])
```

Note however, that the results of these two queries need not compare equal, because we compare queries by the data sent, and the ordering of the data might be different.

Another possibility is to send a collection and a function. In this case, the function is applied to the first few objects of the collection:

```
sage: findmap("Permutations", lambda pi: pi.increasing_tree_shape(), depth=1)
↪# optional -- internet
0: Mp00061 (quality [100])
```

```
>>> from sage.all import *
>>> findmap("Permutations", lambda pi: pi.increasing_tree_shape(),␣
↪depth=Integer(1))      # optional -- internet
0: Mp00061 (quality [100])
```

In rare cases, it may not be possible to guess the codomain of a map, in which case it can be provided as second argument or keyword argument:

```
sage: findmap("Dyck paths", "Perfect matchings", lambda D: [(a+1, b) for a,b in D.
↪tunnels()]) # optional -- internet
0: Mp00146 (quality [100])

sage: findmap("Dyck paths", "Set partitions", lambda D: [(a+1, b) for a,b in D.
↪tunnels()]) # optional -- internet
0: Mp00092oMp00146 (quality [...])
```

```
>>> from sage.all import *
>>> findmap("Dyck paths", "Perfect matchings", lambda D: [(a+Integer(1), b) for a,
↪b in D.tunnels()]) # optional -- internet
0: Mp00146 (quality [100])

>>> findmap("Dyck paths", "Set partitions", lambda D: [(a+Integer(1), b) for a,b␣
↪in D.tunnels()]) # optional -- internet
0: Mp00092oMp00146 (quality [...])
```

Finally, we can also retrieve all maps with a given domain or codomain:

```
sage: findmap("Cc0024")                                                    #␣
↪optional -- internet
Set of combinatorial maps with domain Cc0024: Binary words used by FindStat

sage: findmap(codomain='Cores')                                            #␣
↪optional -- internet
Set of combinatorial maps with codomain Cc0013: Cores used by FindStat
```

```
>>> from sage.all import *
>>> findmap("Cc0024")                                                  # optional -
↪- internet
```

(continues on next page)

(continued from previous page)

```
Set of combinatorial maps with domain Cc0024: Binary words used by FindStat

>>> findmap(codomain='Cores')                                    # optional -
↪- internet
Set of combinatorial maps with codomain Cc0013: Cores used by FindStat
```

sage.databases.findstat.**findstat**(*query=None*, *values=None*, *distribution=None*, *domain=None*, *depth=2*, *max_values=1000*)

Return matching statistics.

INPUT:

One of the following:

- an integer or a string representing a valid FindStat identifier (e.g. 45 or 'St000045'). The keyword arguments `depth` and `max_values` are ignored, `values` and `distribution` must be `None`.
- a list of pairs of the form `(object, value)`, or a dictionary from Sage objects to integer values. The keyword arguments `depth` and `max_values` are passed to the finder, `values` and `distribution` must be `None`.
- a list of pairs of the form (list of objects, list of values), or a single pair of the form (list of objects, list of values). In each pair there should be as many objects as values. The keyword arguments `depth` and `max_values` are passed to the finder.
- a collection and a list of pairs of the form (string, value), or a dictionary from strings to integer values. The keyword arguments `depth` and `max_values` are passed to the finder. This should only be used if the collection is not yet supported.
- a collection and a callable. The callable is used to generate `max_values (object, value)` pairs. The number of terms generated may also be controlled by passing an iterable collection, such as `Permutations(3)`. The keyword arguments `depth` and `max_values` are passed to the finder.

OUTPUT: an instance of a *FindStatStatistic*, represented by

- the FindStat identifier together with its name, or
- a list of triples, each consisting of
  - the statistic
  - a list of strings naming certain maps
  - a number which says how many of the values submitted agree with the values in the database, when applying the maps in the given order to the object and then computing the statistic on the result.

EXAMPLES:

A particular statistic can be retrieved by its St-identifier or number:

```
sage: findstat('St000041')                                          #␣
↪optional -- internet
St000041: The number of nestings of a perfect matching.

sage: findstat(51)                                                  #␣
↪optional -- internet
St000051: The size of the left subtree of a binary tree.

sage: findstat('St000042oMp00116')                                  #␣
```

(continues on next page)

(continued from previous page)

```
↪optional -- internet
St000042oMp00116
```

```
>>> from sage.all import *
>>> findstat('St000041')                                                    # optional -
↪- internet
St000041: The number of nestings of a perfect matching.

>>> findstat(Integer(51))                                                   #␣
↪optional -- internet
St000051: The size of the left subtree of a binary tree.

>>> findstat('St000042oMp00116')                                           # optional -
↪- internet
St000042oMp00116
```

The database can be searched by providing a list of pairs:

```
sage: l = [m for n in range(1, 4) for m in PerfectMatchings(2*n)]
sage: findstat([(m, m.number_of_nestings()) for m in l], depth=0)        #␣
↪optional -- internet
0: St000041 (quality [100, 100])
```

```
>>> from sage.all import *
>>> l = [m for n in range(Integer(1), Integer(4)) for m in␣
↪PerfectMatchings(Integer(2)*n)]
>>> findstat([(m, m.number_of_nestings()) for m in l], depth=Integer(0))        #␣
↪optional -- internet
0: St000041 (quality [100, 100])
```

or a dictionary:

```
sage: findstat({m: m.number_of_nestings() for m in l}, depth=0)          #␣
↪optional -- internet
0: St000041 (quality [100, 100])
```

```
>>> from sage.all import *
>>> findstat({m: m.number_of_nestings() for m in l}, depth=Integer(0))          #␣
↪optional -- internet
0: St000041 (quality [100, 100])
```

Note however, that the results of these two queries need not compare equal, because we compare queries by the data sent, and the ordering of the data might be different.

Another possibility is to send a collection and a function. In this case, the function is applied to the first few objects of the collection:

```
sage: findstat("Perfect Matchings", lambda m: m.number_of_nestings(), depth=0)
↪# optional -- internet
0: St000041 (quality [20, 100])
```

```
>>> from sage.all import *
>>> findstat("Perfect Matchings", lambda m: m.number_of_nestings(),␣
↪depth=Integer(0))    # optional -- internet
0: St000041 (quality [20, 100])
```

To search for a distribution, send a list of lists, or a single pair:

```
sage: PM = PerfectMatchings(10); findstat((PM, [m.number_of_nestings() for m in␣
↪PM]), depth=0) # optional -- internet
0: St000042 (quality [100, 100])
1: St000041 (quality [9, 100])
```

```
>>> from sage.all import *
>>> PM = PerfectMatchings(Integer(10)); findstat((PM, [m.number_of_nestings() for␣
↪m in PM]), depth=Integer(0)) # optional -- internet
0: St000042 (quality [100, 100])
1: St000041 (quality [9, 100])
```

Alternatively, specify the `distribution` parameter:

```
sage: findstat(12, distribution=lambda m: m.number_of_nestings(), depth=0) #␣
↪optional -- internet
0: St000041 (quality [100, 100])
1: St000042 (quality [100, 100])
```

```
>>> from sage.all import *
>>> findstat(Integer(12), distribution=lambda m: m.number_of_nestings(),␣
↪depth=Integer(0)) # optional -- internet
0: St000041 (quality [100, 100])
1: St000042 (quality [100, 100])
```

Note that there is a limit, `FINDSTAT_MAX_VALUES`, on the number of elements that may be submitted to FindStat, which is currently 1000. Therefore, the interface tries to truncate queries appropriately, but this may be impossible, especially with distribution searches:

```
sage: PM = PerfectMatchings(12); PM.cardinality()                    #␣
↪optional -- internet
10395
sage: findstat((PM, [1 for m in PM]))                                #␣
↪optional -- internet
Traceback (most recent call last):
...
ValueError: E016: The statistic finder was unable to perform a search on your␣
↪data. The following errors have occured:

You passed too few elements (0 < 3) to FindStat!
```

```
>>> from sage.all import *
>>> PM = PerfectMatchings(Integer(12)); PM.cardinality()                #␣
↪optional -- internet
10395
>>> findstat((PM, [Integer(1) for m in PM]))                            #␣
```

(continues on next page)

(continued from previous page)

```
↪optional -- internet
Traceback (most recent call last):
...
ValueError: E016: The statistic finder was unable to perform a search on your␣
↪data. The following errors have occured:
<BLANKLINE>
You passed too few elements (0 < 3) to FindStat!
```

Finally, we can also retrieve all statistics with a given domain:

```
sage: findstat("Cc0024")                                                    #␣
↪optional -- internet
Set of combinatorial statistics with domain Cc0024: Binary words in FindStat

sage: findstat(domain='Cores')                                              #␣
↪optional -- internet
Set of combinatorial statistics with domain Cc0013: Cores in FindStat
```

```
>>> from sage.all import *
>>> findstat("Cc0024")                                                  # optional -
↪- internet
Set of combinatorial statistics with domain Cc0024: Binary words in FindStat

>>> findstat(domain='Cores')                                            # optional -
↪- internet
Set of combinatorial statistics with domain Cc0013: Cores in FindStat
```

CHAPTER

TWO

OPTIONAL DATABASES

The following databases require you to install optional packages before full access.

2.1 Cunningham tables

This module provides *cunningham_prime_factors()*, which lists the prime numbers occurring in the factorization of numbers of type b^n+1 or b^n-1 with $b \in \{2,3,5,6,7,10,11,12\}$. For an introduction to Cunningham prime factors, see Wikipedia article Cunningham_Project. The data becomes available if you install the optional cunningham_tables package by the command

```
sage -i cunningham_tables
```

AUTHORS:

- Yann Laigle-Chapuy (2009-10-18): initial version

sage.databases.cunningham_tables.**cunningham_prime_factors**()

List of all the prime numbers occurring in the so called Cunningham table.

They occur in the factorization of numbers of type $b^n + 1$ or $b^n - 1$ with $b \in \{2,3,5,6,7,10,11,12\}$.

EXAMPLES:

```
sage: # optional - cunningham_tables
sage: from sage.databases.cunningham_tables import cunningham_prime_factors
sage: cunningham_prime_factors()
[2,
 3,
 5,
 7,
 11,
 13,
 17,
 ...
```

```
>>> from sage.all import *
>>> # optional - cunningham_tables
>>> from sage.databases.cunningham_tables import cunningham_prime_factors
>>> cunningham_prime_factors()
[2,
 3,
 5,
```

(continues on next page)

(continued from previous page)

```
7,
11,
13,
17,
...
```

2.2 KnotInfo database

This module contains the class *KnotInfoDataBase* and auxiliary classes for it, which serves as an interface to the lists of named knots and links provided at the web-pages KnotInfo and LinkInfo.

To use the database, you need to install the optional database_knotinfo package by the Sage command

```
sage -i database_knotinfo
```

EXAMPLES:

```
sage: # optional - database_knotinfo
sage: from sage.databases.knotinfo_db import KnotInfoDataBase
sage: ki_db = KnotInfoDataBase()
sage: ki_db
<sage.databases.knotinfo_db.KnotInfoDataBase object at ...>
```

```
>>> from sage.all import *
>>> # optional - database_knotinfo
>>> from sage.databases.knotinfo_db import KnotInfoDataBase
>>> ki_db = KnotInfoDataBase()
>>> ki_db
<sage.databases.knotinfo_db.KnotInfoDataBase object at ...>
```

AUTHORS:

- Sebastian Oehms (2020-08): initial version

class sage.databases.knotinfo_db.**KnotInfoColumnTypes**(*value*, *names=<not given>*, **values*, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Bases: Enum

Enum class to specify if a column from the table of knots and links provided at the web-pages KnotInfo and LinkInfo. is used for knots only, links only or both.

EXAMPLES:

```
sage: from sage.databases.knotinfo_db import KnotInfoColumnTypes
sage: [col_type for col_type in KnotInfoColumnTypes]
[<KnotInfoColumnTypes.OnlyKnots: 'K'>,
<KnotInfoColumnTypes.OnlyLinks: 'L'>,
<KnotInfoColumnTypes.KnotsAndLinks: 'B'>]
```

```
>>> from sage.all import *
>>> from sage.databases.knotinfo_db import KnotInfoColumnTypes
>>> [col_type for col_type in KnotInfoColumnTypes]
```

(continues on next page)

(continued from previous page)

```
[<KnotInfoColumnTypes.OnlyKnots: 'K'>,
<KnotInfoColumnTypes.OnlyLinks: 'L'>,
<KnotInfoColumnTypes.KnotsAndLinks: 'B'>]
```

KnotsAndLinks = 'B'

OnlyKnots = 'K'

OnlyLinks = 'L'

class sage.databases.knotinfo_db.**KnotInfoColumns**(*value*, *names=<not given>*, **values*, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Bases: Enum

Enum class to select a column from the table of knots and links provided at the web-pages KnotInfo and LinkInfo.

EXAMPLES:

```
sage: from sage.databases.knotinfo_db import KnotInfoDataBase
sage: ki_db = KnotInfoDataBase()
sage: cols = ki_db.columns(); cols
<enum 'Columns'>
sage: from sage.databases.knotinfo_db import KnotInfoColumns
sage: isinstance(cols.name, KnotInfoColumns)
True

sage: def only_links(c):
....:     return c.column_type() == c.types.OnlyLinks
sage: [c.column_name() for c in cols if only_links(c)]  # optional - database_
↪knotinfo
['Name - Unoriented',
 'Orientation',
 'Unoriented Rank',
 'PD Notation (vector)',
 'PD Notation (KnotTheory)',
 'Braid Notation',
 'Quasipositive Braid',
 'Multivariable Alexander Polynomial',
 'HOMFLYPT Polynomial',
 'Khovanov Polynomial',
 'Unoriented',
 'Arc Notation',
 'Linking Matrix',
 'Rolfsen Name',
 'Components',
 'DT code',
 'Splitting Number',
 'Nullity',
 'Unlinking Number',
 'Weak Splitting Number']
```

```
>>> from sage.all import *
>>> from sage.databases.knotinfo_db import KnotInfoDataBase
>>> ki_db = KnotInfoDataBase()
>>> cols = ki_db.columns(); cols
<enum 'Columns'>
>>> from sage.databases.knotinfo_db import KnotInfoColumns
>>> isinstance(cols.name, KnotInfoColumns)
True

>>> def only_links(c):
...     return c.column_type() == c.types.OnlyLinks
>>> [c.column_name() for c in cols if only_links(c)]  # optional - database_
↪knotinfo
['Name - Unoriented',
 'Orientation',
 'Unoriented Rank',
 'PD Notation (vector)',
 'PD Notation (KnotTheory)',
 'Braid Notation',
 'Quasipositive Braid',
 'Multivariable Alexander Polynomial',
 'HOMFLYPT Polynomial',
 'Khovanov Polynomial',
 'Unoriented',
 'Arc Notation',
 'Linking Matrix',
 'Rolfsen Name',
 'Components',
 'DT code',
 'Splitting Number',
 'Nullity',
 'Unlinking Number',
 'Weak Splitting Number']
```

column_name()

Return the name of `self` displayed on the KnotInfo web-page.

EXAMPLES:

```
sage: from sage.databases.knotinfo_db import KnotInfoDataBase
sage: ki_db = KnotInfoDataBase()
sage: cols = ki_db.columns()
sage: cols.dt_code.column_name()
'DT code'
```

```
>>> from sage.all import *
>>> from sage.databases.knotinfo_db import KnotInfoDataBase
>>> ki_db = KnotInfoDataBase()
>>> cols = ki_db.columns()
>>> cols.dt_code.column_name()
'DT code'
```

column_type()

Return the type of `self`. That is an instance of `Enum` *KnotInfoColumnTypes*.

EXAMPLES:

```
sage: from sage.databases.knotinfo_db import KnotInfoDataBase
sage: ki_db = KnotInfoDataBase()
sage: cols = ki_db.columns()
sage: cols.homfly_polynomial.column_type()
<KnotInfoColumnTypes.OnlyKnots: 'K'>
sage: cols.homflypt_polynomial.column_type()
<KnotInfoColumnTypes.OnlyLinks: 'L'>
sage: cols.name.column_type()
<KnotInfoColumnTypes.KnotsAndLinks: 'B'>
```

```
>>> from sage.all import *
>>> from sage.databases.knotinfo_db import KnotInfoDataBase
>>> ki_db = KnotInfoDataBase()
>>> cols = ki_db.columns()
>>> cols.homfly_polynomial.column_type()
<KnotInfoColumnTypes.OnlyKnots: 'K'>
>>> cols.homflypt_polynomial.column_type()
<KnotInfoColumnTypes.OnlyLinks: 'L'>
>>> cols.name.column_type()
<KnotInfoColumnTypes.KnotsAndLinks: 'B'>
```

description_webpage(*new=0*, *autoraise=True*)

Launch the description page of `self` in the standard web browser.

EXAMPLES:

```
sage: from sage.databases.knotinfo_db import KnotInfoDataBase
sage: ki_db = KnotInfoDataBase()
sage: cols = ki_db.columns()
sage: cols.pd_notation.description_webpage()            # not tested
True
sage: cols.homflypt_polynomial.description_webpage()    # not tested
True
```

```
>>> from sage.all import *
>>> from sage.databases.knotinfo_db import KnotInfoDataBase
>>> ki_db = KnotInfoDataBase()
>>> cols = ki_db.columns()
>>> cols.pd_notation.description_webpage()            # not tested
True
>>> cols.homflypt_polynomial.description_webpage()    # not tested
True
```

property types

Return *KnotInfoColumnTypes* to be used for checks.

EXAMPLES:

```
sage: from sage.databases.knotinfo_db import KnotInfoDataBase
sage: ki_db = KnotInfoDataBase()
sage: cols = ki_db.columns()
```

(continues on next page)

(continued from previous page)

```
sage: cols.dt_code.column_type() == cols.dt_code.types.OnlyLinks
True
```

```
>>> from sage.all import *
>>> from sage.databases.knotinfo_db import KnotInfoDataBase
>>> ki_db = KnotInfoDataBase()
>>> cols = ki_db.columns()
>>> cols.dt_code.column_type() == cols.dt_code.types.OnlyLinks
True
```

class sage.databases.knotinfo_db.**KnotInfoDataBase**(*install=False*)

Bases: SageObject, UniqueRepresentation

Database interface to KnotInfo.

The original data are obtained from KnotInfo web-page (URL see the example below). In order to have these data installed during the build process as a sage-package they are converted as csv files into a tarball. This tarball has been created using the method `create_spkg_tarball()`.

EXAMPLES:

```
sage: from sage.databases.knotinfo_db import KnotInfoDataBase
sage: ki_db = KnotInfoDataBase()
sage: ki_db.filename.knots
<KnotInfoFilename.knots: ['https://knotinfo.math.indiana.edu/',
                          'knotinfo_data_complete']>
```

```
>>> from sage.all import *
>>> from sage.databases.knotinfo_db import KnotInfoDataBase
>>> ki_db = KnotInfoDataBase()
>>> ki_db.filename.knots
<KnotInfoFilename.knots: ['https://knotinfo.math.indiana.edu/',
                          'knotinfo_data_complete']>
```

columns()

Return the columns ot the database table as instances of enum class *KnotInfoColumns*.

EXAMPLES:

```
sage: from sage.databases.knotinfo_db import KnotInfoDataBase
sage: ki_db = KnotInfoDataBase()
sage: cols = ki_db.columns()
sage: [col.column_name() for col in cols if col.name.startswith('pd')]   #␣
↪optional - database_knotinfo
['PD Notation', 'PD Notation (vector)', 'PD Notation (KnotTheory)']
```

```
>>> from sage.all import *
>>> from sage.databases.knotinfo_db import KnotInfoDataBase
>>> ki_db = KnotInfoDataBase()
>>> cols = ki_db.columns()
>>> [col.column_name() for col in cols if col.name.startswith('pd')]   #␣
↪optional - database_knotinfo
['PD Notation', 'PD Notation (vector)', 'PD Notation (KnotTheory)']
```

create_filecache(*force=False*)

Create the internal files containing the database.

INPUT:

- `force` – boolean (default: `False`); if set to `True` the existing file-cache is overwritten

EXAMPLES:

```
sage: from sage.databases.knotinfo_db import KnotInfoDataBase
sage: ki_db = KnotInfoDataBase()
sage: ki_db.create_filecache()    # optional - database_knotinfo
```

```
>>> from sage.all import *
>>> from sage.databases.knotinfo_db import KnotInfoDataBase
>>> ki_db = KnotInfoDataBase()
>>> ki_db.create_filecache()    # optional - database_knotinfo
```

demo_version()

Return whether the KnotInfo databases are installed completely or just the demo version is used.

EXAMPLES:

```
sage: from sage.databases.knotinfo_db import KnotInfoDataBase
sage: ki_db = KnotInfoDataBase()
sage: ki_db.demo_version()       # optional - database_knotinfo
False
```

```
>>> from sage.all import *
>>> from sage.databases.knotinfo_db import KnotInfoDataBase
>>> ki_db = KnotInfoDataBase()
>>> ki_db.demo_version()       # optional - database_knotinfo
False
```

filename

alias of *KnotInfoFilename*

knot_list()

Return the KnotInfo table rows as Python list.

EXAMPLES:

```
sage: from sage.databases.knotinfo_db import KnotInfoDataBase
sage: ki_db = KnotInfoDataBase()
sage: len(ki_db.knot_list())  # not tested (just used on installation)
```

```
>>> from sage.all import *
>>> from sage.databases.knotinfo_db import KnotInfoDataBase
>>> ki_db = KnotInfoDataBase()
>>> len(ki_db.knot_list())  # not tested (just used on installation)
```

link_list()

Return the LinkInfo table rows as Python list.

EXAMPLES:

```
sage: from sage.databases.knotinfo_db import KnotInfoDataBase
sage: ki_db = KnotInfoDataBase()
sage: len(ki_db.link_list())  # not tested (just used on installation)
```

```
>>> from sage.all import *
>>> from sage.databases.knotinfo_db import KnotInfoDataBase
>>> ki_db = KnotInfoDataBase()
>>> len(ki_db.link_list())  # not tested (just used on installation)
```

read(*column*)

Access a column of KnotInfo / LinkInfo.

INPUT:

- `column` – instance of enum *KnotInfoColumns* to select the data to be read in

OUTPUT: a Python list containing the data corresponding to the column

EXAMPLES:

```
sage: from sage.databases.knotinfo_db import KnotInfoDataBase
sage: ki_db = KnotInfoDataBase()
```

```
>>> from sage.all import *
>>> from sage.databases.knotinfo_db import KnotInfoDataBase
>>> ki_db = KnotInfoDataBase()
```

read_column_dict()

Read the dictionary for the column names from the according sobj-file.

OUTPUT: a Python dictionary containing the column names and types

EXAMPLES:

```
sage: from sage.databases.knotinfo_db import KnotInfoDataBase
sage: ki_db = KnotInfoDataBase()
sage: len(ki_db.read_column_dict()) > 120       # optional - database_knotinfo
True
```

```
>>> from sage.all import *
>>> from sage.databases.knotinfo_db import KnotInfoDataBase
>>> ki_db = KnotInfoDataBase()
>>> len(ki_db.read_column_dict()) > Integer(120)       # optional - database_
↪knotinfo
True
```

read_num_knots()

Read the number of knots contained in the database (without proper links) from the according sobj-file.

OUTPUT: integer

EXAMPLES:

```
sage: from sage.databases.knotinfo_db import KnotInfoDataBase
sage: ki_db = KnotInfoDataBase()
```

(continues on next page)

(continued from previous page)

```
sage: ki_db.read_num_knots()              # optional - database_knotinfo
12966
```

```
>>> from sage.all import *
>>> from sage.databases.knotinfo_db import KnotInfoDataBase
>>> ki_db = KnotInfoDataBase()
>>> ki_db.read_num_knots()              # optional - database_knotinfo
12966
```

read_row_dict()

Read the dictionary for the row names that is the knot and link names from the according sobj-file

OUTPUT:

A python dictionary containing the names of the knots and links together with their table index and the corresponding number of components

EXAMPLES:

```
sage: from sage.databases.knotinfo_db import KnotInfoDataBase
sage: ki_db = KnotInfoDataBase()
sage: ki_db.read_row_dict()['K7_1']
[8, 1]
```

```
>>> from sage.all import *
>>> from sage.databases.knotinfo_db import KnotInfoDataBase
>>> ki_db = KnotInfoDataBase()
>>> ki_db.read_row_dict()['K7_1']
[8, 1]
```

row_names()

Return a dictionary to obtain the original name to a row_dict key.

OUTPUT: a Python dictionary containing the names of the knots and links together with their original names from the database

EXAMPLES:

```
sage: from sage.databases.knotinfo_db import KnotInfoDataBase
sage: ki_db = KnotInfoDataBase()
sage: ki_db.row_names()['K7_1']       # optional - database_knotinfo
'7_1'
```

```
>>> from sage.all import *
>>> from sage.databases.knotinfo_db import KnotInfoDataBase
>>> ki_db = KnotInfoDataBase()
>>> ki_db.row_names()['K7_1']       # optional - database_knotinfo
'7_1'
```

version()

Return the version of the database currently installed on the device.

Note

The development of the original databases on the KnotInfo and LinkInfo web-pages is in a continuous flow. The installed version can be behind the current available state of these databases. Every month a cronjob on the GitHub repository searches for differences and creates a new release on PyPI in case of success.

If you note that your version is behind the version on PyPI and would like to have Sage working with that release you should first try to upgrade using `sage -i database_knotinfo`. If this is not successful even though you are on the latest Sage release please create an issue for that in the GitHub repository.

EXAMPLES:

```
sage: from sage.databases.knotinfo_db import KnotInfoDataBase
sage: ki_db = KnotInfoDataBase()
sage: ki_db.version()  >= '2021.10.1'  # optional database_knotinfo
True
```

```
>>> from sage.all import *
>>> from sage.databases.knotinfo_db import KnotInfoDataBase
>>> ki_db = KnotInfoDataBase()
>>> ki_db.version()  >= '2021.10.1'  # optional database_knotinfo
True
```

class sage.databases.knotinfo_db.**KnotInfoFilename**(*value*, *names=<not given>*, **values*, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Bases: Enum

Enum for the different data files. The following choices are possible:

- `knots` – contains the data from KnotInfo
- `links` – contains the data for proper links from LinkInfo

Examples:

```
sage: from sage.databases.knotinfo_db import KnotInfoDataBase
sage: ki_db = KnotInfoDataBase()
sage: ki_db.filename
<enum 'KnotInfoFilename'>
```

```
>>> from sage.all import *
>>> from sage.databases.knotinfo_db import KnotInfoDataBase
>>> ki_db = KnotInfoDataBase()
>>> ki_db.filename
<enum 'KnotInfoFilename'>
```

csv()

Return the file name under which the data from the web-page are stored as csv file.

Examples:

```
sage: from sage.databases.knotinfo_db import KnotInfoDataBase
sage: ki_db = KnotInfoDataBase()
```

(continues on next page)

(continued from previous page)

```
sage: ki_db.filename.knots.csv()
'knotinfo_data_complete.csv'
```

```
>>> from sage.all import *
>>> from sage.databases.knotinfo_db import KnotInfoDataBase
>>> ki_db = KnotInfoDataBase()
>>> ki_db.filename.knots.csv()
'knotinfo_data_complete.csv'
```

description_url(*column*)

Return the url of the description page of the given column.

Examples:

```
sage: from sage.databases.knotinfo_db import KnotInfoDataBase
sage: ki_db = KnotInfoDataBase()
sage: ki_db.filename.knots.description_url(ki_db.columns().braid_notation)
'https://knotinfo.math.indiana.edu/descriptions/braid_notation.html'
```

```
>>> from sage.all import *
>>> from sage.databases.knotinfo_db import KnotInfoDataBase
>>> ki_db = KnotInfoDataBase()
>>> ki_db.filename.knots.description_url(ki_db.columns().braid_notation)
'https://knotinfo.math.indiana.edu/descriptions/braid_notation.html'
```

diagram_url(*fname*, *single=False*)

Return the url of the diagram page of the given link.

Examples:

```
sage: from sage.databases.knotinfo_db import KnotInfoDataBase
sage: ki_db = KnotInfoDataBase()
sage: ki_db.filename.knots.diagram_url('3_1-50.png')
'https://knotinfo.math.indiana.edu/diagram_display.php?3_1-50.png'
sage: ki_db.filename.knots.diagram_url('3_1', single=True)
'https://knotinfo.math.indiana.edu/diagrams/3_1'
```

```
>>> from sage.all import *
>>> from sage.databases.knotinfo_db import KnotInfoDataBase
>>> ki_db = KnotInfoDataBase()
>>> ki_db.filename.knots.diagram_url('3_1-50.png')
'https://knotinfo.math.indiana.edu/diagram_display.php?3_1-50.png'
>>> ki_db.filename.knots.diagram_url('3_1', single=True)
'https://knotinfo.math.indiana.edu/diagrams/3_1'
```

excel()

Return the Excel-file name to download the data from the web-page.

Examples:

```
sage: from sage.databases.knotinfo_db import KnotInfoDataBase
sage: ki_db = KnotInfoDataBase()
```

(continues on next page)

(continued from previous page)

```
sage: ki_db.filename.knots.excel()
'knotinfo_data_complete.xls'
```

```
>>> from sage.all import *
>>> from sage.databases.knotinfo_db import KnotInfoDataBase
>>> ki_db = KnotInfoDataBase()
>>> ki_db.filename.knots.excel()
'knotinfo_data_complete.xls'
```

knots = ['https://knotinfo.math.indiana.edu/', 'knotinfo_data_complete']

links = ['https://linkinfo.sitehost.iu.edu/', 'linkinfo_data_complete']

num_knots(*version*)

Return the file name under which the number of knots is stored in an sobj-file.

Examples:

```
sage: from sage.databases.knotinfo_db import KnotInfoDataBase
sage: ki_db = KnotInfoDataBase()
sage: ki_db.filename.knots.num_knots('21.7')
'num_knots_21.7.sobj'
```

```
>>> from sage.all import *
>>> from sage.databases.knotinfo_db import KnotInfoDataBase
>>> ki_db = KnotInfoDataBase()
>>> ki_db.filename.knots.num_knots('21.7')
'num_knots_21.7.sobj'
```

sobj_column()

Return the file name under which the column-data of the csv-File is stored as python dictionary in a sobj-file.

Examples:

```
sage: from sage.databases.knotinfo_db import KnotInfoDataBase
sage: ki_db = KnotInfoDataBase()
sage: ki_db.filename.knots.sobj_column()
'column_dict.sobj'
```

```
>>> from sage.all import *
>>> from sage.databases.knotinfo_db import KnotInfoDataBase
>>> ki_db = KnotInfoDataBase()
>>> ki_db.filename.knots.sobj_column()
'column_dict.sobj'
```

sobj_data(*column*)

Return the file name under which the data of the given column is stored as python list in a sobj-file.

Examples:

```
sage: from sage.databases.knotinfo_db import KnotInfoDataBase
sage: ki_db = KnotInfoDataBase()
sage: ki_db.filename.knots.sobj_data(ki_db.columns().braid_notation)
'knotinfo_braid_notation'
```

```
>>> from sage.all import *
>>> from sage.databases.knotinfo_db import KnotInfoDataBase
>>> ki_db = KnotInfoDataBase()
>>> ki_db.filename.knots.sobj_data(ki_db.columns().braid_notation)
'knotinfo_braid_notation'
```

sobj_row()

Return the file name under which the row-data of the csv-File is stored as python dictionary in a sobj-file.

Examples:

```
sage: from sage.databases.knotinfo_db import KnotInfoDataBase
sage: ki_db = KnotInfoDataBase()
sage: ki_db.filename.knots.sobj_row()
'row_dict.sobj'
```

```
>>> from sage.all import *
>>> from sage.databases.knotinfo_db import KnotInfoDataBase
>>> ki_db = KnotInfoDataBase()
>>> ki_db.filename.knots.sobj_row()
'row_dict.sobj'
```

url()

Return the URL to download the data from the web-page.

Examples:

```
sage: from sage.databases.knotinfo_db import KnotInfoDataBase
sage: ki_db = KnotInfoDataBase()
sage: ki_db.filename.knots.url()
'https://knotinfo.math.indiana.edu/'
```

```
>>> from sage.all import *
>>> from sage.databases.knotinfo_db import KnotInfoDataBase
>>> ki_db = KnotInfoDataBase()
>>> ki_db.filename.knots.url()
'https://knotinfo.math.indiana.edu/'
```

class sage.databases.knotinfo_db.**dc**(*value*, *names=<not given>*, **values*, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Bases: *KnotInfoColumns*

alexander_polynomial = ['Alexander', KnotInfoColumnTypes.OnlyKnots]

alternating = ['Alternating', KnotInfoColumnTypes.KnotsAndLinks]

arc_notation = ['Arc Notation', KnotInfoColumnTypes.OnlyLinks]

braid_index = ['Braid Index', KnotInfoColumnTypes.OnlyKnots]

braid_length = ['Braid Length', KnotInfoColumnTypes.OnlyKnots]

braid_notation = ['Braid Notation', KnotInfoColumnTypes.KnotsAndLinks]

braid_notation_old = ['Braid Notation', KnotInfoColumnTypes.OnlyLinks]

```
conway_polynomial = ['Conway', KnotInfoColumnTypes.KnotsAndLinks]

cosmetic_crossing = ['Cosmetic Crossing', KnotInfoColumnTypes.OnlyKnots]

crossing_number = ['Crossing Number', KnotInfoColumnTypes.KnotsAndLinks]

determinant = ['Determinant', KnotInfoColumnTypes.KnotsAndLinks]

dt_code = ['DT code', KnotInfoColumnTypes.OnlyLinks]

dt_notation = ['DT Notation', KnotInfoColumnTypes.OnlyKnots]

fibered = ['Fibered', KnotInfoColumnTypes.OnlyKnots]

gauss_notation = ['Gauss Notation', KnotInfoColumnTypes.KnotsAndLinks]

geometric_type = ['Geometric Type', KnotInfoColumnTypes.OnlyKnots]

homfly_polynomial = ['HOMFLY', KnotInfoColumnTypes.OnlyKnots]

homflypt_polynomial = ['HOMFLYPT Polynomial', KnotInfoColumnTypes.OnlyLinks]

jones_polynomial = ['Jones', KnotInfoColumnTypes.KnotsAndLinks]

kauffman_polynomial = ['Kauffman', KnotInfoColumnTypes.KnotsAndLinks]

khovanov_odd_integral_polynomial = ['KH Odd Red Z Poly',
KnotInfoColumnTypes.OnlyKnots]

khovanov_odd_mod2_polynomial = ['KH Red Odd Mod2 Poly',
KnotInfoColumnTypes.OnlyKnots]

khovanov_odd_rational_polynomial = ['KH Odd Red Q Poly',
KnotInfoColumnTypes.OnlyKnots]

khovanov_polynomial = ['Khovanov', KnotInfoColumnTypes.OnlyLinks]

khovanov_reduced_integral_polynomial = ['KH Red Z Poly',
KnotInfoColumnTypes.OnlyKnots]

khovanov_reduced_mod2_polynomial = ['KH Red Mod2 Poly',
KnotInfoColumnTypes.OnlyKnots]

khovanov_reduced_rational_polynomial = ['KH Red Q Poly',
KnotInfoColumnTypes.OnlyKnots]

khovanov_unreduced_integral_polynomial = ['KH Unred Z Poly',
KnotInfoColumnTypes.OnlyKnots]

name = ['Name', KnotInfoColumnTypes.KnotsAndLinks]

name_unoriented = ['Name - Unoriented', KnotInfoColumnTypes.OnlyLinks]

pd_notation = ['PD Notation', KnotInfoColumnTypes.OnlyKnots]

pd_notation_vector = ['PD Notation (vector)', KnotInfoColumnTypes.OnlyLinks]

positive = ['Positive', KnotInfoColumnTypes.OnlyKnots]
```

```
symmetry_type = ['Symmetry Type', KnotInfoColumnTypes.OnlyKnots]

unoriented = ['Unoriented', KnotInfoColumnTypes.OnlyLinks]

width = ['Width', KnotInfoColumnTypes.OnlyKnots]
```

2.3 Cubic Hecke database

This module contains the class *CubicHeckeDataBase* which serves as an interface to Ivan Marin's data files with respect to the cubic Hecke algebras. The data is available via a Python wrapper as a pip installable package database_cubic_hecke. For installation hints please see the documentation there. All data needed for the cubic Hecke algebras on less than four strands is included in this module for demonstration purpose (see for example *read_basis()*, *read_irr()* , ... generated with the help of *create_demo_data()*).

In addition to Ivan Marin's data the package contains a function *read_markov()* to obtain the coefficients of Markov traces on the cubic Hecke algebras. This data has been precomputed with the help of `create_markov_trace_data.py` in the database_cubic_hecke repository. Again, for less than four strands, this data is includes here for demonstration purposes.

Furthermore, this module contains the class *CubicHeckeFileCache* that enables `CubicHeckeAlgebra` to keep intermediate results of calculations in the file system.

The enum *MarkovTraceModuleBasis* serves as basis for the submodule of linear forms on the cubic Hecke algebra on at most four strands satisfying the Markov trace condition for its cubic Hecke subalgebras.

To use the database, you need to install the optional database_cubic_hecke package by the Sage command

```
sage -i database_cubic_hecke
```

EXAMPLES:

```
sage: # optional - database_cubic_hecke
sage: from sage.databases.cubic_hecke_db import CubicHeckeDataBase
sage: cha_db = CubicHeckeDataBase()
sage: cha_db
<sage.databases.cubic_hecke_db.CubicHeckeDataBase object at ...>
```

```
>>> from sage.all import *
>>> # optional - database_cubic_hecke
>>> from sage.databases.cubic_hecke_db import CubicHeckeDataBase
>>> cha_db = CubicHeckeDataBase()
>>> cha_db
<sage.databases.cubic_hecke_db.CubicHeckeDataBase object at ...>
```

AUTHORS:

- Sebastian Oehms (2020-05): initial version
- Sebastian Oehms (2022-03): PyPi version and Markov trace functionality

class sage.databases.cubic_hecke_db.**CubicHeckeDataBase**

Bases: `SageObject`

Database interface for `CubicHeckeAlgebra`

The original data are obtained from Ivan Marin's web page

The data needed to work with the cubic Hecke algebras on less than 4 strands is completely contained in this module. Data needed for the larger algebras can be installed as an optional Sage package which comes as a `pip` installable Python wrapper of Ivan Marin's data. For more information see the corresponding repository.

EXAMPLES:

```
sage: from sage.databases.cubic_hecke_db import CubicHeckeDataBase
sage: cha_db = CubicHeckeDataBase()
sage: cha_db._feature
Feature('database_cubic_hecke')
```

```
>>> from sage.all import *
>>> from sage.databases.cubic_hecke_db import CubicHeckeDataBase
>>> cha_db = CubicHeckeDataBase()
>>> cha_db._feature
Feature('database_cubic_hecke')
```

demo_version()

Return whether the cubic Hecke database is installed completely or just the demo version is used.

EXAMPLES:

```
sage: from sage.databases.knotinfo_db import KnotInfoDataBase
sage: ki_db = KnotInfoDataBase()
sage: ki_db.demo_version()        # optional - database_knotinfo
False
```

```
>>> from sage.all import *
>>> from sage.databases.knotinfo_db import KnotInfoDataBase
>>> ki_db = KnotInfoDataBase()
>>> ki_db.demo_version()        # optional - database_knotinfo
False
```

read(*section*, *variables=None*, *nstrands=4*)

Access various static data libraries.

INPUT:

- `section` – instance of enum *CubicHeckeDataSection* to select the data to be read in

OUTPUT: a dictionary containing the data corresponding to the section

EXAMPLES:

```
sage: from sage.databases.cubic_hecke_db import CubicHeckeDataBase
sage: cha_db = CubicHeckeDataBase()
sage: basis = cha_db.read(cha_db.section.basis, nstrands=3)
sage: len(basis)
24
```

```
>>> from sage.all import *
>>> from sage.databases.cubic_hecke_db import CubicHeckeDataBase
>>> cha_db = CubicHeckeDataBase()
>>> basis = cha_db.read(cha_db.section.basis, nstrands=Integer(3))
>>> len(basis)
24
```

read_matrix_representation(*representation_type*, *gen_ind*, *nstrands*, *ring_of_definition*)

Return the matrix representations from the database.

INPUT:

- `representation_type` – an element of `RepresentationType` specifying the type of the representation

EXAMPLES:

```
sage: from sage.databases.cubic_hecke_db import CubicHeckeDataBase
sage: CHA3 = algebras.CubicHecke(2)
sage: GER = CHA3.extension_ring(generic=True)
sage: cha_db = CHA3._database
sage: rt = CHA3.repr_type
sage: m1 =cha_db.read_matrix_representation(rt.SplitIrredMarin, 1, 3, GER)
sage: len(m1)
7
sage: GBR = CHA3.base_ring(generic=True)
sage: m1rl = cha_db.read_matrix_representation(rt.RegularLeft, 1, 3, GBR)
sage: m1rl[0].dimensions()
(24, 24)
```

```
>>> from sage.all import *
>>> from sage.databases.cubic_hecke_db import CubicHeckeDataBase
>>> CHA3 = algebras.CubicHecke(Integer(2))
>>> GER = CHA3.extension_ring(generic=True)
>>> cha_db = CHA3._database
>>> rt = CHA3.repr_type
>>> m1 =cha_db.read_matrix_representation(rt.SplitIrredMarin, Integer(1),␣
↪Integer(3), GER)
>>> len(m1)
7
>>> GBR = CHA3.base_ring(generic=True)
>>> m1rl = cha_db.read_matrix_representation(rt.RegularLeft, Integer(1),␣
↪Integer(3), GBR)
>>> m1rl[Integer(0)].dimensions()
(24, 24)
```

section

alias of *CubicHeckeDataSection*

version()

Return the current version of the database.

EXAMPLES:

```
sage: from sage.databases.cubic_hecke_db import CubicHeckeDataBase
sage: cha_db = CubicHeckeDataBase()
sage: cha_db.version() > '2022.1.1'    # optional - database_cubic_hecke
True
```

```
>>> from sage.all import *
>>> from sage.databases.cubic_hecke_db import CubicHeckeDataBase
>>> cha_db = CubicHeckeDataBase()
```

(continues on next page)

(continued from previous page)

```
>>> cha_db.version() > '2022.1.1'    # optional - database_cubic_hecke
True
```

class sage.databases.cubic_hecke_db.**CubicHeckeDataSection**(*value*, *names=<not given>*, **values*, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Bases: Enum

Enum for the different sections of the database.

The following choices are possible:

- `basis` – list of basis elements
- `reg_left_reprs` – data for the left regular representation
- `reg_right_reprs` – data for the right regular representation
- `irr_reprs` – data for the split irreducible representations
- `markov_tr_cfs` – data for the coefficients of the formal Markov traces

EXAMPLES:

```
sage: from sage.databases.cubic_hecke_db import CubicHeckeDataBase
sage: cha_db = CubicHeckeDataBase()
sage: cha_db.section
<enum 'CubicHeckeDataSection'>
```

```
>>> from sage.all import *
>>> from sage.databases.cubic_hecke_db import CubicHeckeDataBase
>>> cha_db = CubicHeckeDataBase()
>>> cha_db.section
<enum 'CubicHeckeDataSection'>
```

basis = 'basis'

markov_tr_cfs = 'markov_tr_cfs'

regular_left = 'regular_left'

regular_right = 'regular_right'

split_irred = 'split_irred'

class sage.databases.cubic_hecke_db.**CubicHeckeFileCache**(*num_strands*)

Bases: SageObject

A class to cache calculations of `CubicHeckeAlgebra` in the local file system.

is_empty(*section=None*)

Return `True` if the cache of the given `section` is empty.

INPUT:

- `section` – an element of *CubicHeckeFileCache.section* to select the section of the file cache or `None` (default) meaning all sections

EXAMPLES:

```
sage: from sage.databases.cubic_hecke_db import CubicHeckeFileCache
sage: cha2_fc = CubicHeckeFileCache(2)
sage: cha2_fc.reset_library()
sage: cha2_fc.is_empty()
True
```

```
>>> from sage.all import *
>>> from sage.databases.cubic_hecke_db import CubicHeckeFileCache
>>> cha2_fc = CubicHeckeFileCache(Integer(2))
>>> cha2_fc.reset_library()
>>> cha2_fc.is_empty()
True
```

read(*section*)

Read data into memory from the file system.

INPUT:

- `section` – an element of *CubicHeckeFileCache.section* specifying the section where the corresponding cached data belong to

OUTPUT:

Dictionary containing the data library corresponding to the section of file cache

EXAMPLES:

```
sage: from sage.databases.cubic_hecke_db import CubicHeckeFileCache
sage: cha2_fc = CubicHeckeFileCache(2)
sage: cha2_fc.reset_library(cha2_fc.section.braid_images)
sage: cha2_fc.read(cha2_fc.section.braid_images)
{}
```

```
>>> from sage.all import *
>>> from sage.databases.cubic_hecke_db import CubicHeckeFileCache
>>> cha2_fc = CubicHeckeFileCache(Integer(2))
>>> cha2_fc.reset_library(cha2_fc.section.braid_images)
>>> cha2_fc.read(cha2_fc.section.braid_images)
{}
```

read_braid_image(*braid_tietze*, *ring_of_definition*)

Return the list of pre calculated braid images from file cache.

INPUT:

- `braid_tietze` – tuple representing the braid in Tietze form
- `ring_of_definition` – a `CubicHeckeRingOfDefinition`

OUTPUT:

A dictionary containing the pre calculated braid image of the given braid.

EXAMPLES:

```
sage: from sage.databases.cubic_hecke_db import CubicHeckeFileCache
sage: CHA2 = algebras.CubicHecke(2)
```

(continues on next page)

(continued from previous page)

```
sage: ring_of_definition = CHA2.base_ring(generic=True)
sage: cha_fc = CubicHeckeFileCache(2)
sage: B2 = BraidGroup(2)
sage: b, = B2.gens(); b2 = b**2
sage: cha_fc.is_empty(CubicHeckeFileCache.section.braid_images)
True
sage: b2_img = CHA2(b2); b2_img
w*c^-1 + u*c - v
sage: cha_fc.write_braid_image(b2.Tietze(), b2_img.to_vector())
sage: cha_fc.read_braid_image(b2.Tietze(), ring_of_definition)
(-v, u, w)
```

```
>>> from sage.all import *
>>> from sage.databases.cubic_hecke_db import CubicHeckeFileCache
>>> CHA2 = algebras.CubicHecke(Integer(2))
>>> ring_of_definition = CHA2.base_ring(generic=True)
>>> cha_fc = CubicHeckeFileCache(Integer(2))
>>> B2 = BraidGroup(Integer(2))
>>> b, = B2.gens(); b2 = b**Integer(2)
>>> cha_fc.is_empty(CubicHeckeFileCache.section.braid_images)
True
>>> b2_img = CHA2(b2); b2_img
w*c^-1 + u*c - v
>>> cha_fc.write_braid_image(b2.Tietze(), b2_img.to_vector())
>>> cha_fc.read_braid_image(b2.Tietze(), ring_of_definition)
(-v, u, w)
```

read_matrix_representation(*representation_type*, *monomial_tietze*, *ring_of_definition*)

Return the matrix representations of the given monomial (in Tietze form) if it has been stored in the file cache before. INPUT:

- `representation_type` – an element of `RepresentationType` specifying the type of the representation
- `monomial_tietze` – tuple representing the braid in Tietze form
- `ring_of_definition` – instance of `CubicHeckeRingOfDefinition` respectively `CubicHeckeExtensionRing` depending on whether `representation_type` is split or not

OUTPUT:

Dictionary containing all matrix representations of `self` of the given `representation_type` which have been stored in the file cache. Otherwise `None` is returned.

EXAMPLES:

```
sage: CHA2 = algebras.CubicHecke(2)
sage: R = CHA2.base_ring(generic=True)
sage: cha_fc = CHA2._filecache
sage: g, = CHA2.gens(); gt = g.Tietze()
sage: rt = CHA2.repr_type
sage: g.matrix(representation_type=rt.RegularLeft)
[ 0 -v  1]
[ 1  u  0]
[ 0  w  0]
```

(continues on next page)

(continued from previous page)

```
sage: [_] == cha_fc.read_matrix_representation(rt.RegularLeft, gt, R)
True
sage: cha_fc.reset_library(cha_fc.section.matrix_representations)
sage: cha_fc.write(cha_fc.section.matrix_representations)
sage: cha_fc.read_matrix_representation(rt.RegularLeft, gt, R) == None
True
```

```
>>> from sage.all import *
>>> CHA2 = algebras.CubicHecke(Integer(2))
>>> R = CHA2.base_ring(generic=True)
>>> cha_fc = CHA2._filecache
>>> g, = CHA2.gens(); gt = g.Tietze()
>>> rt = CHA2.repr_type
>>> g.matrix(representation_type=rt.RegularLeft)
[ 0 -v  1]
[ 1  u  0]
[ 0  w  0]
>>> [_] == cha_fc.read_matrix_representation(rt.RegularLeft, gt, R)
True
>>> cha_fc.reset_library(cha_fc.section.matrix_representations)
>>> cha_fc.write(cha_fc.section.matrix_representations)
>>> cha_fc.read_matrix_representation(rt.RegularLeft, gt, R) == None
True
```

reset_library(*section=None*)

Reset the file cache corresponding to the specified `section`.

INPUT:

- `section` – an element of *CubicHeckeFileCache.section* to select the section of the file cache or `None` (default) meaning all sections

EXAMPLES:

```
sage: from sage.databases.cubic_hecke_db import CubicHeckeFileCache
sage: cha2_fc = CubicHeckeFileCache(2)
sage: cha2_fc.reset_library(cha2_fc.section.braid_images)
sage: cha2_fc.read(cha2_fc.section.braid_images)
{}
sage: cha2_fc.reset_library(cha2_fc.section.matrix_representations)
sage: data_mat = cha2_fc.read(cha2_fc.section.matrix_representations)
sage: len(data_mat.keys())
4
```

```
>>> from sage.all import *
>>> from sage.databases.cubic_hecke_db import CubicHeckeFileCache
>>> cha2_fc = CubicHeckeFileCache(Integer(2))
>>> cha2_fc.reset_library(cha2_fc.section.braid_images)
>>> cha2_fc.read(cha2_fc.section.braid_images)
{}
>>> cha2_fc.reset_library(cha2_fc.section.matrix_representations)
>>> data_mat = cha2_fc.read(cha2_fc.section.matrix_representations)
>>> len(data_mat.keys())
```

(continues on next page)

(continued from previous page)

```
4
```

class section(*value*, *names=<not given>*, **values*, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Bases: Enum

Enum for the different sections of file cache. The following choices are possible:

- matrix_representations – file cache for representation matrices of basis elements
- braid_images – file cache for images of braids
- basis_extensions – file cache for a dynamical growing basis used in the case of cubic Hecke algebras on more than 4 strands
- markov_trace – file cache for intermediate results of long calculations in order to recover the results already obtained by previous attemps of calculation until the corresponding intermediate step

EXAMPLES:

```
sage: from sage.databases.cubic_hecke_db import CubicHeckeFileCache
sage: CHA2 = algebras.CubicHecke(2)
sage: cha_fc = CubicHeckeFileCache(CHA2)
sage: cha_fc.section
<enum 'section'>
```

```
>>> from sage.all import *
>>> from sage.databases.cubic_hecke_db import CubicHeckeFileCache
>>> CHA2 = algebras.CubicHecke(Integer(2))
>>> cha_fc = CubicHeckeFileCache(CHA2)
>>> cha_fc.section
<enum 'section'>
```

basis_extensions = 'basis_extensions'

braid_images = 'braid_images'

filename(*nstrands=None*)

Return the file name under which the data of this file cache section is stored as an sobj-file.

INPUT:

- nstrands – (optional) Integer; number of strands of the underlying braid group if the data file depends on it

EXAMPLES:

```
sage: from sage.databases.cubic_hecke_db import CubicHeckeFileCache
sage: CHA2 = algebras.CubicHecke(2)
sage: cha_fc = CubicHeckeFileCache(CHA2)
sage: cha_fc.section.matrix_representations.filename(2)
'matrix_representations_2.sobj'
sage: cha_fc.section.braid_images.filename(2)
'braid_images_2.sobj'
```

```
>>> from sage.all import *
>>> from sage.databases.cubic_hecke_db import CubicHeckeFileCache
```

(continues on next page)

(continued from previous page)

```
>>> CHA2 = algebras.CubicHecke(Integer(2))
>>> cha_fc = CubicHeckeFileCache(CHA2)
>>> cha_fc.section.matrix_representations.filename(Integer(2))
'matrix_representations_2.sobj'
>>> cha_fc.section.braid_images.filename(Integer(2))
'braid_images_2.sobj'
```

markov_trace = 'markov_trace'

matrix_representations = 'matrix_representations'

update_basis_extensions(*new_basis_extensions*)

Update the file cache for basis extensions for cubic Hecke algebras on more than 4 strands according to the given `new_basis_extensions`.

INPUT:

- `new_basis_extensions` – list of additional (to the static basis) basis elements which should replace the former such list in the file

EXAMPLES:

```
sage: from sage.databases.cubic_hecke_db import CubicHeckeFileCache
sage: CHA2 = algebras.CubicHecke(2)
sage: cha_fc = CubicHeckeFileCache(2)
sage: cha_fc.is_empty(CubicHeckeFileCache.section.basis_extensions)
True
sage: cha_fc.update_basis_extensions([(1,), (-1,)])
sage: cha_fc.read(CubicHeckeFileCache.section.basis_extensions)
[(1,), (-1,)]
sage: cha_fc.reset_library(CubicHeckeFileCache.section.basis_extensions)
sage: cha_fc.write(CubicHeckeFileCache.section.basis_extensions)
sage: cha_fc.is_empty(CubicHeckeFileCache.section.basis_extensions)
True
```

```
>>> from sage.all import *
>>> from sage.databases.cubic_hecke_db import CubicHeckeFileCache
>>> CHA2 = algebras.CubicHecke(Integer(2))
>>> cha_fc = CubicHeckeFileCache(Integer(2))
>>> cha_fc.is_empty(CubicHeckeFileCache.section.basis_extensions)
True
>>> cha_fc.update_basis_extensions([(Integer(1),), (-Integer(1),)])
>>> cha_fc.read(CubicHeckeFileCache.section.basis_extensions)
[(1,), (-1,)]
>>> cha_fc.reset_library(CubicHeckeFileCache.section.basis_extensions)
>>> cha_fc.write(CubicHeckeFileCache.section.basis_extensions)
>>> cha_fc.is_empty(CubicHeckeFileCache.section.basis_extensions)
True
```

write(*section=None*)

Write data from memory to the file system.

INPUT:

- `section` – an element of *CubicHeckeFileCache.section* specifying the section where the corresponding cached data belong to; if omitted, the data of all sections is written to the file system

EXAMPLES:

```
sage: from sage.databases.cubic_hecke_db import CubicHeckeFileCache
sage: cha2_fc = CubicHeckeFileCache(2)
sage: cha2_fc.reset_library(cha2_fc.section.braid_images)
sage: cha2_fc.write(cha2_fc.section.braid_images)
```

```
>>> from sage.all import *
>>> from sage.databases.cubic_hecke_db import CubicHeckeFileCache
>>> cha2_fc = CubicHeckeFileCache(Integer(2))
>>> cha2_fc.reset_library(cha2_fc.section.braid_images)
>>> cha2_fc.write(cha2_fc.section.braid_images)
```

write_braid_image(*braid_tietze*, *braid_image_vect*)

Write the braid image of the given braid to the file cache.

INPUT:

- `braid_tietze` – tuple representing the braid in Tietze form
- `braid_image_vect` – image of the given braid as a vector with respect to the basis of the cubic Hecke algebra

EXAMPLES:

```
sage: from sage.databases.cubic_hecke_db import CubicHeckeFileCache
sage: CHA2 = algebras.CubicHecke(2)
sage: ring_of_definition = CHA2.base_ring(generic=True)
sage: cha_fc = CubicHeckeFileCache(2)
sage: B2 = BraidGroup(2)
sage: b, = B2.gens(); b3 = b**3
sage: b3_img = CHA2(b3); b3_img
u*w*c^-1 + (u^2-v)*c - (u*v-w)
sage: cha_fc.write_braid_image(b3.Tietze(), b3_img.to_vector())
sage: cha_fc.read_braid_image(b3.Tietze(), ring_of_definition)
(-u*v + w, u^2 - v, u*w)
sage: cha_fc.reset_library(CubicHeckeFileCache.section.braid_images)
sage: cha_fc.write(CubicHeckeFileCache.section.braid_images)
sage: cha_fc.is_empty(CubicHeckeFileCache.section.braid_images)
True
```

```
>>> from sage.all import *
>>> from sage.databases.cubic_hecke_db import CubicHeckeFileCache
>>> CHA2 = algebras.CubicHecke(Integer(2))
>>> ring_of_definition = CHA2.base_ring(generic=True)
>>> cha_fc = CubicHeckeFileCache(Integer(2))
>>> B2 = BraidGroup(Integer(2))
>>> b, = B2.gens(); b3 = b**Integer(3)
>>> b3_img = CHA2(b3); b3_img
u*w*c^-1 + (u^2-v)*c - (u*v-w)
>>> cha_fc.write_braid_image(b3.Tietze(), b3_img.to_vector())
>>> cha_fc.read_braid_image(b3.Tietze(), ring_of_definition)
(-u*v + w, u^2 - v, u*w)
>>> cha_fc.reset_library(CubicHeckeFileCache.section.braid_images)
>>> cha_fc.write(CubicHeckeFileCache.section.braid_images)
```

(continues on next page)

(continued from previous page)

```
>>> cha_fc.is_empty(CubicHeckeFileCache.section.braid_images)
True
```

write_matrix_representation(*representation_type*, *monomial_tietze*, *matrix_list*)

Write the matrix representation of a monomial to the file cache.

INPUT:

- `representation_type` – an element of `RepresentationType` specifying the type of the representation
- `monomial_tietze` – tuple representing the braid in Tietze form
- `matrix_list` – list of matrices corresponding to the irreducible representations

EXAMPLES:

```
sage: CHA2 = algebras.CubicHecke(2)
sage: R = CHA2.base_ring(generic=True)
sage: cha_fc = CHA2._filecache
sage: g, = CHA2.gens(); gi = ~g; git = gi.Tietze()
sage: rt = CHA2.repr_type
sage: m = gi.matrix(representation_type=rt.RegularRight)
sage: cha_fc.read_matrix_representation(rt.RegularRight, git, R)
[
[     0      1 (-u)/w]
[     0      0    1/w]
[     1      0    v/w]
]
sage: CHA2.reset_filecache(cha_fc.section.matrix_representations)
sage: cha_fc.read_matrix_representation(rt.RegularLeft, git, R) == None
True
sage: cha_fc.write_matrix_representation(rt.RegularRight, git, [m])
sage: [m] == cha_fc.read_matrix_representation(rt.RegularRight, git, R)
True
```

```
>>> from sage.all import *
>>> CHA2 = algebras.CubicHecke(Integer(2))
>>> R = CHA2.base_ring(generic=True)
>>> cha_fc = CHA2._filecache
>>> g, = CHA2.gens(); gi = ~g; git = gi.Tietze()
>>> rt = CHA2.repr_type
>>> m = gi.matrix(representation_type=rt.RegularRight)
>>> cha_fc.read_matrix_representation(rt.RegularRight, git, R)
[
[     0      1 (-u)/w]
[     0      0    1/w]
[     1      0    v/w]
]
>>> CHA2.reset_filecache(cha_fc.section.matrix_representations)
>>> cha_fc.read_matrix_representation(rt.RegularLeft, git, R) == None
True
>>> cha_fc.write_matrix_representation(rt.RegularRight, git, [m])
>>> [m] == cha_fc.read_matrix_representation(rt.RegularRight, git, R)
```

(continues on next page)

(continued from previous page)

```
True
```

class sage.databases.cubic_hecke_db.**MarkovTraceModuleBasis**(*value*, *names=<not given>*, **values*, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Bases: Enum

Enum for the basis elements for the Markov trace module.

The choice of the basis elements doesn't have a systematically background apart from generating the submodule of maximal rank in the module of linear forms on the cubic Hecke algebra for which the Markov trace condition with respect to its cubic Hecke subalgebras hold. The number of crossings in the corresponding links is chosen as minimal as possible.

EXAMPLES:

```
sage: from sage.databases.cubic_hecke_db import MarkovTraceModuleBasis
sage: MarkovTraceModuleBasis.K92.description()
'knot 9_34'
```

```
>>> from sage.all import *
>>> from sage.databases.cubic_hecke_db import MarkovTraceModuleBasis
>>> MarkovTraceModuleBasis.K92.description()
'knot 9_34'
```

```
K4 = ['knot 4_1', 3, (1, -2, 1, -2), None]

K4U = ['knot 4_1 plus one unlink', 4, (1, -2, 1, -2), [[3, 8, 4, 9], [9, 7, 10, 6],
[7, 4, 8, 5], [5, 11, 6, 10], [11, 1, 12, 2], [12, 1, 3, 2]]]

K6 = ['knot 6_1', 4, (1, 1, 2, -1, -3, 2, -3), None]

K7 = ['knot 7_4', 4, (1, 1, 2, -1, 2, 2, 3, -2, 3), None]

K91 = ['knot 9_29', 4, (1, -2, -2, 3, -2, 1, -2, 3, -2), None]

K92 = ['knot 9_34', 4, (-1, 2, -1, 2, -3, 2, -1, 2, -3), None]

U1 = ['one unlink', 1, (), []]

U2 = ['two unlinks', 2, (), [[3, 1, 4, 2], [4, 1, 3, 2]]]

U3 = ['three unlinks', 3, (), [[3, 7, 4, 8], [4, 7, 5, 8], [5, 1, 6, 2], [6, 1, 3,
2]]]

U4 = ['four unlinks', 4, (), [[3, 9, 4, 10], [4, 9, 5, 10], [5, 11, 6, 12], [6, 11,
7, 12], [7, 1, 8, 2], [8, 1, 3, 2]]]
```

braid_tietze(*strands_embed=None*)

Return the Tietze representation of the braid corresponding to this basis element.

INPUT:

- `strands_embed` – (optional) the number of strands of the braid if strands should be added

OUTPUT: a tuple representing the braid in Tietze form

EXAMPLES:

```
sage: from sage.databases.cubic_hecke_db import MarkovTraceModuleBasis
sage: MarkovTraceModuleBasis.U2.braid_tietze()
()
sage: MarkovTraceModuleBasis.U2.braid_tietze(strands_embed=4)
(2, 3)
```

```
>>> from sage.all import *
>>> from sage.databases.cubic_hecke_db import MarkovTraceModuleBasis
>>> MarkovTraceModuleBasis.U2.braid_tietze()
()
>>> MarkovTraceModuleBasis.U2.braid_tietze(strands_embed=Integer(4))
(2, 3)
```

description()

Return a description of the link corresponding to this basis element.

In the case of knots it refers to the naming according to KnotInfo.

EXAMPLES:

```
sage: from sage.databases.cubic_hecke_db import MarkovTraceModuleBasis
sage: MarkovTraceModuleBasis.U3.description()
'three unlinks'
```

```
>>> from sage.all import *
>>> from sage.databases.cubic_hecke_db import MarkovTraceModuleBasis
>>> MarkovTraceModuleBasis.U3.description()
'three unlinks'
```

link()

Return the Link that represents this basis element.

EXAMPLES:

```
sage: from sage.databases.cubic_hecke_db import MarkovTraceModuleBasis
sage: MarkovTraceModuleBasis.U1.link()
Link with 1 component represented by 0 crossings
sage: MarkovTraceModuleBasis.K4.link()
Link with 1 component represented by 4 crossings
```

```
>>> from sage.all import *
>>> from sage.databases.cubic_hecke_db import MarkovTraceModuleBasis
>>> MarkovTraceModuleBasis.U1.link()
Link with 1 component represented by 0 crossings
>>> MarkovTraceModuleBasis.K4.link()
Link with 1 component represented by 4 crossings
```

links_gould_polynomial()

Return the Links-Gould polynomial of the link that represents this basis element.

EXAMPLES:

```
sage: from sage.databases.cubic_hecke_db import MarkovTraceModuleBasis
sage: MarkovTraceModuleBasis.U1.links_gould_polynomial()
```

(continues on next page)

(continued from previous page)

```
1
sage: MarkovTraceModuleBasis.U2.links_gould_polynomial()
0
sage: MarkovTraceModuleBasis.K4.links_gould_polynomial()
2*t0*t1 - 3*t0 - 3*t1 + t0*t1^-1 + 7 + t0^-1*t1
- 3*t1^-1 - 3*t0^-1 + 2*t0^-1*t1^-1
```

```
>>> from sage.all import *
>>> from sage.databases.cubic_hecke_db import MarkovTraceModuleBasis
>>> MarkovTraceModuleBasis.U1.links_gould_polynomial()
1
>>> MarkovTraceModuleBasis.U2.links_gould_polynomial()
0
>>> MarkovTraceModuleBasis.K4.links_gould_polynomial()
2*t0*t1 - 3*t0 - 3*t1 + t0*t1^-1 + 7 + t0^-1*t1
- 3*t1^-1 - 3*t0^-1 + 2*t0^-1*t1^-1
```

regular_homfly_polynomial()

Return the regular variant of the HOMFLY-PT polynomial of the link that represents this basis element.

This is the HOMFLY-PT polynomial renormalized by the writhe factor such that it is an invariant of regular isotopy.

EXAMPLES:

```
sage: from sage.databases.cubic_hecke_db import MarkovTraceModuleBasis
sage: MarkovTraceModuleBasis.U1.regular_homfly_polynomial()
1
sage: u2 = MarkovTraceModuleBasis.U2.regular_homfly_polynomial(); u2
-L*M^-1 - L^-1*M^-1
sage: u2**2 == MarkovTraceModuleBasis.U3.regular_homfly_polynomial()
True
sage: u2**3 == MarkovTraceModuleBasis.U4.regular_homfly_polynomial()
True
```

```
>>> from sage.all import *
>>> from sage.databases.cubic_hecke_db import MarkovTraceModuleBasis
>>> MarkovTraceModuleBasis.U1.regular_homfly_polynomial()
1
>>> u2 = MarkovTraceModuleBasis.U2.regular_homfly_polynomial(); u2
-L*M^-1 - L^-1*M^-1
>>> u2**Integer(2) == MarkovTraceModuleBasis.U3.regular_homfly_polynomial()
True
>>> u2**Integer(3) == MarkovTraceModuleBasis.U4.regular_homfly_polynomial()
True
```

regular_kauffman_polynomial()

Return the regular variant of the Kauffman polynomial of the link that represents this basis element.

This is the Kauffman polynomial renormalized by the writhe factor such that it is an invariant of regular isotopy.

EXAMPLES:

```
sage: from sage.databases.cubic_hecke_db import MarkovTraceModuleBasis
sage: MarkovTraceModuleBasis.U1.regular_homfly_polynomial()
1
sage: u2 = MarkovTraceModuleBasis.U2.regular_kauffman_polynomial(); u2
a*z^-1 - 1 + a^-1*z^-1
sage: u2**2 == MarkovTraceModuleBasis.U3.regular_kauffman_polynomial()
True
sage: u2**3 == MarkovTraceModuleBasis.U4.regular_kauffman_polynomial()
True
```

```
>>> from sage.all import *
>>> from sage.databases.cubic_hecke_db import MarkovTraceModuleBasis
>>> MarkovTraceModuleBasis.U1.regular_homfly_polynomial()
1
>>> u2 = MarkovTraceModuleBasis.U2.regular_kauffman_polynomial(); u2
a*z^-1 - 1 + a^-1*z^-1
>>> u2**Integer(2) == MarkovTraceModuleBasis.U3.regular_kauffman_polynomial()
True
>>> u2**Integer(3) == MarkovTraceModuleBasis.U4.regular_kauffman_polynomial()
True
```

strands()

Return the number of strands of the minimal braid representative of the link corresponding to `self`.

EXAMPLES:

```
sage: from sage.databases.cubic_hecke_db import MarkovTraceModuleBasis
sage: MarkovTraceModuleBasis.K7.strands()
4
```

```
>>> from sage.all import *
>>> from sage.databases.cubic_hecke_db import MarkovTraceModuleBasis
>>> MarkovTraceModuleBasis.K7.strands()
4
```

writhe()

Return the writhe of the link corresponding to this basis element.

EXAMPLES:

```
sage: from sage.databases.cubic_hecke_db import MarkovTraceModuleBasis
sage: MarkovTraceModuleBasis.K4.writhe()
0
sage: MarkovTraceModuleBasis.K6.writhe()
1
```

```
>>> from sage.all import *
>>> from sage.databases.cubic_hecke_db import MarkovTraceModuleBasis
>>> MarkovTraceModuleBasis.K4.writhe()
0
>>> MarkovTraceModuleBasis.K6.writhe()
1
```

sage.databases.cubic_hecke_db.**create_demo_data**(*filename='demo_data.py'*)

Generate code for the functions inserted below to access the small cases of less than 4 strands as demonstration cases.

The code is written to a file with the given name from where it can be copied into this source file (in case a change is needed).

EXAMPLES:

```
sage: from sage.databases.cubic_hecke_db import create_demo_data
sage: create_demo_data()    # not tested
```

```
>>> from sage.all import *
>>> from sage.databases.cubic_hecke_db import create_demo_data
>>> create_demo_data()    # not tested
```

sage.databases.cubic_hecke_db.**read_basis**(*num_strands=3*)

Return precomputed data of Ivan Marin.

This code was generated by *create_demo_data()*, please do not edit.

INPUT:

- num_strands – integer; number of strands of the cubic Hecke algebra

EXAMPLES:

```
sage: from sage.databases.cubic_hecke_db import read_basis
sage: read_basis(2)
[[], [1], [-1]]
```

```
>>> from sage.all import *
>>> from sage.databases.cubic_hecke_db import read_basis
>>> read_basis(Integer(2))
[[], [1], [-1]]
```

sage.databases.cubic_hecke_db.**read_irr**(*variables*, *num_strands=3*)

Return precomputed data of Ivan Marin.

This code was generated by *create_demo_data()*, please do not edit.

INPUT:

- variables – tuple containing the indeterminates of the representation
- num_strands – integer; number of strands of the cubic Hecke algebra

EXAMPLES:

```
sage: from sage.databases.cubic_hecke_db import read_irr
sage: L.<a, b, c, j> = LaurentPolynomialRing(ZZ)
sage: read_irr((a, b, c, j), 2)
([1, 1, 1],
[[{(0, 0): a}], [{(0, 0): c}], [{(0, 0): b}]],
[[{(0, 0): a^-1}], [{(0, 0): c^-1}], [{(0, 0): b^-1}]])
```

```
>>> from sage.all import *
>>> from sage.databases.cubic_hecke_db import read_irr
>>> L = LaurentPolynomialRing(ZZ, names=('a', 'b', 'c', 'j',)); (a, b, c, j,) = L.
↪_first_ngens(4)
>>> read_irr((a, b, c, j), Integer(2))
([1, 1, 1],
[[{(0, 0): a}], [{(0, 0): c}], [{(0, 0): b}]],
[[{(0, 0): a^-1}], [{(0, 0): c^-1}], [{(0, 0): b^-1}]])
```

sage.databases.cubic_hecke_db.**read_markov**(*bas_ele*, *variables*, *num_strands=4*)

Return precomputed Markov trace coefficients.

This code was generated by `create_markov_trace_data.py` (from the `database_cubic_hecke` repository), please do not edit.

INPUT:

- `bas_ele` – an element of *MarkovTraceModuleBasis*
- `variables` – tuple consisting of the variables used in the coefficients
- `num_strands` – integer (default: 4); the number of strands

OUTPUT:

A list of the coefficients. The i'th member corresponds to the i'th basis element.

EXAMPLES:

```
sage: # needs sympy
sage: from sage.databases.cubic_hecke_db import read_markov
sage: from sympy import var
sage: u, v, w, s = var('u, v, w, s')
sage: variables = (u, v, w, s)
sage: read_markov('U2', variables, num_strands=3)
[0, s, 1/s, s, 1/s, 0, 0, 0, 0, -s*v, s, s, -s*u/w, -v/s, 1/s,
 0, 0, 0, 0, 1/s, -u/(s*w), -v/s, 0, 0]
```

```
>>> from sage.all import *
>>> # needs sympy
>>> from sage.databases.cubic_hecke_db import read_markov
>>> from sympy import var
>>> u, v, w, s = var('u, v, w, s')
>>> variables = (u, v, w, s)
>>> read_markov('U2', variables, num_strands=Integer(3))
[0, s, 1/s, s, 1/s, 0, 0, 0, 0, -s*v, s, s, -s*u/w, -v/s, 1/s,
 0, 0, 0, 0, 1/s, -u/(s*w), -v/s, 0, 0]
```

sage.databases.cubic_hecke_db.**read_regl**(*variables*, *num_strands=3*)

Return precomputed data of Ivan Marin.

This code was generated by *create_demo_data()*, please do not edit.

INPUT:

- `variables` – tuple containing the indeterminates of the representation
- `num_strands` – integer; number of strands of the cubic Hecke algebra

EXAMPLES:

```
sage: from sage.databases.cubic_hecke_db import read_regl
sage: L.<u, v, w> = LaurentPolynomialRing(ZZ)
sage: read_regl((u, v, w), 2)
([3],
[[{(0, 1): -v, (0, 2): 1, (1, 0): 1, (1, 1): u, (2, 1): w}]],
[[{(0, 1): 1, (0, 2): -u*w^-1, (1, 2): w^-1, (2, 0): 1, (2, 2): v*w^-1}]])
```

```
>>> from sage.all import *
>>> from sage.databases.cubic_hecke_db import read_regl
>>> L = LaurentPolynomialRing(ZZ, names=('u', 'v', 'w',)); (u, v, w,) = L._first_
↪ngens(3)
>>> read_regl((u, v, w), Integer(2))
([3],
[[{(0, 1): -v, (0, 2): 1, (1, 0): 1, (1, 1): u, (2, 1): w}]],
[[{(0, 1): 1, (0, 2): -u*w^-1, (1, 2): w^-1, (2, 0): 1, (2, 2): v*w^-1}]])
```

sage.databases.cubic_hecke_db.**read_regr**(*variables*, *num_strands=3*)

Return precomputed data of Ivan Marin.

This code was generated by *create_demo_data()*, please do not edit.

INPUT:

- variables – tuple containing the indeterminates of the representation
- num_strands – integer; number of strands of the cubic Hecke algebra

EXAMPLES:

```
sage: from sage.databases.cubic_hecke_db import read_regr
sage: L.<u, v, w> = LaurentPolynomialRing(ZZ)
sage: read_regr((u, v, w), 2)
([3],
[[{(0, 1): -v, (0, 2): 1, (1, 0): 1, (1, 1): u, (2, 1): w}]],
[[{(0, 1): 1, (0, 2): -u*w^-1, (1, 2): w^-1, (2, 0): 1, (2, 2): v*w^-1}]])
```

```
>>> from sage.all import *
>>> from sage.databases.cubic_hecke_db import read_regr
>>> L = LaurentPolynomialRing(ZZ, names=('u', 'v', 'w',)); (u, v, w,) = L._first_
↪ngens(3)
>>> read_regr((u, v, w), Integer(2))
([3],
[[{(0, 1): -v, (0, 2): 1, (1, 0): 1, (1, 1): u, (2, 1): w}]],
[[{(0, 1): 1, (0, 2): -u*w^-1, (1, 2): w^-1, (2, 0): 1, (2, 2): v*w^-1}]])
```

sage.databases.cubic_hecke_db.**simplify**(*mat*)

Convert a matrix to a dictionary consisting of flat Python objects.

INPUT:

- mat – matrix to be converted into a dict

OUTPUT:

A dict from which mat can be reconstructed via element construction. The values of the dictionary may be dictionaries of tuples of integers or strings.

EXAMPLES:

```
sage: from sage.databases.cubic_hecke_db import simplify
sage: import sage.algebras.hecke_algebras.cubic_hecke_base_ring as chbr
sage: ER.<a, b, c> = chbr.CubicHeckeExtensionRing()
sage: mat = matrix(ER, [[2*a, -3], [c, 4*b*~c]]); mat
[     2*a       -3]
[       c 4*b*c^-1]
sage: simplify(mat)
{(0, 0): {(1, 0, 0): {0: 2}},
 (0, 1): {(0, 0, 0): {0: -3}},
 (1, 0): {(0, 0, 1): {0: 1}},
 (1, 1): {(0, 1, -1): {0: 4}}}
sage: mat == matrix(ER, _)
True
sage: F = ER.fraction_field()
sage: matf = mat.change_ring(F)
sage: simplify(matf)
{(0, 0): '2*a', (0, 1): '-3', (1, 0): 'c', (1, 1): '4*b/c'}
sage: matf == matrix(F, _)
True
```

```
>>> from sage.all import *
>>> from sage.databases.cubic_hecke_db import simplify
>>> import sage.algebras.hecke_algebras.cubic_hecke_base_ring as chbr
>>> ER = chbr.CubicHeckeExtensionRing(names=('a', 'b', 'c',)); (a, b, c,) = ER._
↪first_ngens(3)
>>> mat = matrix(ER, [[Integer(2)*a, -Integer(3)], [c, Integer(4)*b*~c]]); mat
[     2*a       -3]
[       c 4*b*c^-1]
>>> simplify(mat)
{(0, 0): {(1, 0, 0): {0: 2}},
 (0, 1): {(0, 0, 0): {0: -3}},
 (1, 0): {(0, 0, 1): {0: 1}},
 (1, 1): {(0, 1, -1): {0: 4}}}
>>> mat == matrix(ER, _)
True
>>> F = ER.fraction_field()
>>> matf = mat.change_ring(F)
>>> simplify(matf)
{(0, 0): '2*a', (0, 1): '-3', (1, 0): 'c', (1, 1): '4*b/c'}
>>> matf == matrix(F, _)
True
```

2.4 Ideals from the Symbolic Data project

This module implements a thin wrapper for the optional symbolic dataset of ideals as published on http://www.symbolicdata.org. From the project website:

> For different purposes, algorithms and implementations are tested on certified and reliable data. The development of tools and data for such tests is usually 'orthogonal' to the main implementation efforts, it requires different skills and technologies and is not loved by programmers. On the other hand, in many cases, tools and data could easily be reused - with slight modifications - across similar projects. The SymbolicData

Project is set out to coordinate such efforts within the Computer Algebra Community. Commonly collected certified and reliable data can also be used to compare otherwise incomparable approaches, algorithms, and implementations. Benchmark suites and Challenges for symbolic computations are not as well established as in other areas of computer science. This is probably due to the fact that there are not yet well agreed aims of such a benchmarking. Nevertheless various (often high quality) special benchmarks are scattered through the literature. During the last years, efforts toward collection of test data for symbolic computations were intensified. They focused mainly on the creation of general benchmarks for different areas of symbolic computation and the collection of such activities on different Web site. For further qualification of these efforts, it would be of great benefit to create a commonly available digital archive of these special benchmark data scattered through the literature. This would provide the community with an electronic repository of certified data that could be addressed and extended during further development.

In order to use this dataset, you need to install the optional database_symbolic_data package by the Sage command

```
sage -i database_symbolic_data
```

EXAMPLES:

```
sage: # optional - database_symbolic_data
sage: sd = SymbolicData(); sd
SymbolicData with 372 ideals
sage: sd.ZeroDim__example_1
Ideal (x1^2 + x2^2 - 10, x1^2 + x1*x2 + 2*x2^2 - 16) of Multivariate Polynomial Ring␣
↪in x1, x2 over Rational Field
sage: sd.Katsura_3
Ideal (u0 + 2*u1 + 2*u2 + 2*u3 - 1,
       u1^2 + 2*u0*u2 + 2*u1*u3 - u2,
       2*u0*u1 + 2*u1*u2 + 2*u2*u3 - u1,
       u0^2 + 2*u1^2 + 2*u2^2 + 2*u3^2 - u0) of Multivariate Polynomial Ring in u0,␣
↪u1, u2, u3 over Rational Field
sage: sd.get_ideal('Katsura_3', GF(127), 'degrevlex')
Ideal (u0 + 2*u1 + 2*u2 + 2*u3 - 1,
       u1^2 + 2*u0*u2 + 2*u1*u3 - u2,
       2*u0*u1 + 2*u1*u2 + 2*u2*u3 - u1,
       u0^2 + 2*u1^2 + 2*u2^2 + 2*u3^2 - u0) of Multivariate Polynomial Ring in u0,␣
↪u1, u2, u3 over Finite Field of size 127
```

```
>>> from sage.all import *
>>> # optional - database_symbolic_data
>>> sd = SymbolicData(); sd
SymbolicData with 372 ideals
>>> sd.ZeroDim__example_1
Ideal (x1^2 + x2^2 - 10, x1^2 + x1*x2 + 2*x2^2 - 16) of Multivariate Polynomial Ring␣
↪in x1, x2 over Rational Field
>>> sd.Katsura_3
Ideal (u0 + 2*u1 + 2*u2 + 2*u3 - 1,
       u1^2 + 2*u0*u2 + 2*u1*u3 - u2,
       2*u0*u1 + 2*u1*u2 + 2*u2*u3 - u1,
       u0^2 + 2*u1^2 + 2*u2^2 + 2*u3^2 - u0) of Multivariate Polynomial Ring in u0,␣
↪u1, u2, u3 over Rational Field
>>> sd.get_ideal('Katsura_3', GF(Integer(127)), 'degrevlex')
Ideal (u0 + 2*u1 + 2*u2 + 2*u3 - 1,
       u1^2 + 2*u0*u2 + 2*u1*u3 - u2,
       2*u0*u1 + 2*u1*u2 + 2*u2*u3 - u1,
```

(continues on next page)

(continued from previous page)

```
      u0^2 + 2*u1^2 + 2*u2^2 + 2*u3^2 - u0) of Multivariate Polynomial Ring in u0,␣
↪u1, u2, u3 over Finite Field of size 127
```

AUTHORS:

- Martin Albrecht (2007-02-19): initial version

class sage.databases.symbolic_data.**SymbolicData**

Bases: object

Database of ideals as distributed by The SymbolicData Project (http://symbolicdata.org).

This class needs the optional database_symbolic_data package to be installed.

get_ideal(*name*, *base_ring=Rational Field*, *term_order='degrevlex'*)

Return the ideal given by 'name' over the base ring given by 'base_ring' in a polynomial ring with the term order given by 'term_order'.

INPUT:

- name – name as on the symbolic data website
- base_ring – base ring for the polynomial ring (default: QQ)
- term_order – term order for the polynomial ring (default: degrevlex)

OUTPUT: ideal as given by name in PolynomialRing(base_ring,vars,term_order)

EXAMPLES:

```
sage: sd = SymbolicData() # optional - database_symbolic_data
sage: sd.get_ideal('Katsura_3',GF(127),'degrevlex') # optional - database_
↪symbolic_data
Ideal (u0 + 2*u1 + 2*u2 + 2*u3 - 1,
       u1^2 + 2*u0*u2 + 2*u1*u3 - u2,
       2*u0*u1 + 2*u1*u2 + 2*u2*u3 - u1,
       u0^2 + 2*u1^2 + 2*u2^2 + 2*u3^2 - u0) of Multivariate Polynomial Ring␣
↪in u0, u1, u2, u3 over Finite Field of size 127
```

```
>>> from sage.all import *
>>> sd = SymbolicData() # optional - database_symbolic_data
>>> sd.get_ideal('Katsura_3',GF(Integer(127)),'degrevlex') # optional -␣
↪database_symbolic_data
Ideal (u0 + 2*u1 + 2*u2 + 2*u3 - 1,
       u1^2 + 2*u0*u2 + 2*u1*u3 - u2,
       2*u0*u1 + 2*u1*u2 + 2*u2*u3 - u1,
       u0^2 + 2*u1^2 + 2*u2^2 + 2*u3^2 - u0) of Multivariate Polynomial Ring␣
↪in u0, u1, u2, u3 over Finite Field of size 127
```

2.5 Cremona's tables of elliptic curves

Sage includes John Cremona's tables of elliptic curves in an easy-to-use format. An instance of the class CremonaDatabase() gives access to the database.

If the optional full CremonaDatabase is not installed, a mini-version is included by default with Sage. It contains Weierstrass equations, rank, and torsion for curves up to conductor 10000.

The large database includes all curves in John Cremona's tables. It also includes data related to the BSD conjecture and modular degrees for all of these curves, and generators for the Mordell-Weil groups. To install it via the optional database_cremona_ellcurve package, run the following command in the shell

```
sage -i database_cremona_ellcurve
```

This causes the latest version of the database to be downloaded from the internet.

Both the mini and full versions of John Cremona's tables are stored in SAGE_SHARE/cremona as SQLite databases. The mini version has the layout:

```
CREATE TABLE t_class(conductor INTEGER, class TEXT PRIMARY KEY, rank INTEGER);
CREATE TABLE t_curve(class TEXT, curve TEXT PRIMARY KEY, eqn TEXT UNIQUE, tors␣
↪INTEGER);
CREATE INDEX i_t_class_conductor ON t_class(conductor);
CREATE INDEX i_t_curve_class ON t_curve(class);
```

while the full version has the layout:

```
CREATE TABLE t_class(conductor INTEGER, class TEXT PRIMARY KEY, rank INTEGER, L REAL,␣
↪deg INTEGER);
CREATE TABLE t_curve(class TEXT, curve TEXT PRIMARY KEY, eqn TEXT UNIQUE, gens TEXT,␣
↪tors INTEGER, cp INTEGER, om REAL, reg REAL, sha);
CREATE INDEX i_t_class_conductor ON t_class(conductor);
CREATE INDEX i_t_curve_class ON t_curve(class);
```

sage.databases.cremona.**CremonaDatabase**(*name=None*, *mini=None*)

Initialize the Cremona database with name `name`.

If `name` is `None` it instead initializes large Cremona database (named 'cremona'), if available or default mini Cremona database (named 'cremona mini').

If the Cremona database in question is in the format of the mini database, you must set `mini=True`, otherwise it must be set to `False`.

class sage.databases.cremona.**LargeCremonaDatabase**(*name*, *read_only=True*, *build=False*)

Bases: *MiniCremonaDatabase*

The Cremona database of elliptic curves.

EXAMPLES:

```
sage: c = CremonaDatabase('cremona')  # optional - database_cremona_ellcurve
sage: c.allcurves(11)                 # optional - database_cremona_ellcurve
{'a1': [[0, -1, 1, -10, -20], 0, 5],
'a2': [[0, -1, 1, -7820, -263580], 0, 1],
'a3': [[0, -1, 1, 0, 0], 0, 5]}
```

```
>>> from sage.all import *
>>> c = CremonaDatabase('cremona')  # optional - database_cremona_ellcurve
>>> c.allcurves(Integer(11))                # optional - database_cremona_
↪ellcurve
{'a1': [[0, -1, 1, -10, -20], 0, 5],
'a2': [[0, -1, 1, -7820, -263580], 0, 1],
'a3': [[0, -1, 1, 0, 0], 0, 5]}
```

allbsd(N)

Return the allbsd table for conductor N. The entries are:

```
[id, tamagawa_product, Omega_E, L, Reg_E, Sha_an(E)]
```

where id is the isogeny class (letter) followed by a number, e.g., b3, and L is $L^r(E,1)/r!$, where E has rank r.

INPUT:

- N – integer; the conductor

OUTPUT: dictionary containing the allbsd table for each isogeny class in conductor N

EXAMPLES:

```
sage: c = CremonaDatabase()
sage: c.allbsd(12)            # optional - database_cremona_ellcurve
{}
sage: c.allbsd(19)['a3']      # optional - database_cremona_ellcurve
[1, 4.07927920046493, 0.453253244496104, 1.0, 1]
sage: c.allbsd(12001)['a1']   # optional - database_cremona_ellcurve
[2, 3.27608135248722, 1.54910143090506, 0.236425971187952, 1.0]
```

```
>>> from sage.all import *
>>> c = CremonaDatabase()
>>> c.allbsd(Integer(12))            # optional - database_cremona_ellcurve
{}
>>> c.allbsd(Integer(19))['a3']      # optional - database_cremona_ellcurve
[1, 4.07927920046493, 0.453253244496104, 1.0, 1]
>>> c.allbsd(Integer(12001))['a1']   # optional - database_cremona_ellcurve
[2, 3.27608135248722, 1.54910143090506, 0.236425971187952, 1.0]
```

allgens(N)

Return the allgens table for conductor N.

INPUT:

- N – integer; the conductor

OUTPUT: dictionary; id:[points, …], …

EXAMPLES:

```
sage: c = CremonaDatabase()
sage: c.allgens(12)            # optional - database_cremona_ellcurve
{}
sage: c.allgens(1001)['a1']    # optional - database_cremona_ellcurve
[[61, 181, 1]]
sage: c.allgens(12001)['a1']   # optional - database_cremona_ellcurve
[[7, 2, 1]]
```

```
>>> from sage.all import *
>>> c = CremonaDatabase()
>>> c.allgens(Integer(12))            # optional - database_cremona_ellcurve
{}
>>> c.allgens(Integer(1001))['a1']    # optional - database_cremona_ellcurve
```

(continues on next page)

(continued from previous page)

```
[[61, 181, 1]]
>>> c.allgens(Integer(12001))['a1']    # optional - database_cremona_ellcurve
[[7, 2, 1]]
```

degphi(*N*)

Return the degphi table for conductor N.

INPUT:

- N – integer; the conductor

OUTPUT: dictionary; id:degphi, …

EXAMPLES:

```
sage: c = CremonaDatabase()
sage: c.degphi(11)             # optional - database_cremona_ellcurve
{'a1': 1}
sage: c.degphi(12001)['c1']    # optional - database_cremona_ellcurve
1640
```

```
>>> from sage.all import *
>>> c = CremonaDatabase()
>>> c.degphi(Integer(11))             # optional - database_cremona_ellcurve
{'a1': 1}
>>> c.degphi(Integer(12001))['c1']    # optional - database_cremona_ellcurve
1640
```

class sage.databases.cremona.**MiniCremonaDatabase**(*name*, *read_only=True*, *build=False*)

Bases: SQLDatabase

The Cremona database of elliptic curves.

EXAMPLES:

```
sage: c = CremonaDatabase()
sage: c.allcurves(11)
{'a1': [[0, -1, 1, -10, -20], 0, 5],
 'a2': [[0, -1, 1, -7820, -263580], 0, 1],
 'a3': [[0, -1, 1, 0, 0], 0, 5]}
```

```
>>> from sage.all import *
>>> c = CremonaDatabase()
>>> c.allcurves(Integer(11))
{'a1': [[0, -1, 1, -10, -20], 0, 5],
 'a2': [[0, -1, 1, -7820, -263580], 0, 1],
 'a3': [[0, -1, 1, 0, 0], 0, 5]}
```

allcurves(*N*)

Return the allcurves table of curves of conductor N.

INPUT:

- N – integer; the conductor

OUTPUT: dictionary; id:[ainvs, rank, tor], …

EXAMPLES:

```
sage: c = CremonaDatabase()
sage: c.allcurves(11)['a3']
[[0, -1, 1, 0, 0], 0, 5]
sage: c.allcurves(12)
{}
sage: c.allcurves(12001)['a1']    # optional - database_cremona_ellcurve
[[1, 0, 0, -101, 382], 1, 1]
```

```
>>> from sage.all import *
>>> c = CremonaDatabase()
>>> c.allcurves(Integer(11))['a3']
[[0, -1, 1, 0, 0], 0, 5]
>>> c.allcurves(Integer(12))
{}
>>> c.allcurves(Integer(12001))['a1']    # optional - database_cremona_ellcurve
[[1, 0, 0, -101, 382], 1, 1]
```

coefficients_and_data(*label*)

Return the Weierstrass coefficients and other data for the curve with given label.

EXAMPLES:

```
sage: c, d = CremonaDatabase().coefficients_and_data('144b1')
sage: c
[0, 0, 0, 6, 7]
sage: d['conductor']
144
sage: d['cremona_label']
'144b1'
sage: d['rank']
0
sage: d['torsion_order']
2
```

```
>>> from sage.all import *
>>> c, d = CremonaDatabase().coefficients_and_data('144b1')
>>> c
[0, 0, 0, 6, 7]
>>> d['conductor']
144
>>> d['cremona_label']
'144b1'
>>> d['rank']
0
>>> d['torsion_order']
2
```

Check that Issue #17904 is fixed:

```
sage: 'gens' in CremonaDatabase().coefficients_and_data('100467a2')[1] #␣
↪optional - database_cremona_ellcurve
True
```

```
>>> from sage.all import *
>>> 'gens' in CremonaDatabase().coefficients_and_data('100467a2')[Integer(1)]
↪# optional - database_cremona_ellcurve
True
```

conductor_range()

Return the range of conductors that are covered by the database.

OUTPUT: tuple of ints (N1,N2+1) where N1 is the smallest and N2 the largest conductor for which the database is complete.

EXAMPLES:

```
sage: c = CremonaDatabase('cremona mini')
sage: c.conductor_range()
(1, 10000)
```

```
>>> from sage.all import *
>>> c = CremonaDatabase('cremona mini')
>>> c.conductor_range()
(1, 10000)
```

curves(*N*)

Return the curves table of all *optimal* curves of conductor N.

INPUT:

- N – integer; the conductor

OUTPUT: dictionary; id:[ainvs, rank, tor], …

EXAMPLES:

Optimal curves of conductor 37:

```
sage: CremonaDatabase().curves(37)
{'a1': [[0, 0, 1, -1, 0], 1, 1], 'b1': [[0, 1, 1, -23, -50], 0, 3]}
```

```
>>> from sage.all import *
>>> CremonaDatabase().curves(Integer(37))
{'a1': [[0, 0, 1, -1, 0], 1, 1], 'b1': [[0, 1, 1, -23, -50], 0, 3]}
```

Note the 'h3', which is the unique case in the tables where the optimal curve doesn't have label ending in 1:

```
sage: sorted(CremonaDatabase().curves(990))
['a1', 'b1', 'c1', 'd1', 'e1', 'f1', 'g1', 'h3', 'i1', 'j1', 'k1', 'l1']
```

```
>>> from sage.all import *
>>> sorted(CremonaDatabase().curves(Integer(990)))
['a1', 'b1', 'c1', 'd1', 'e1', 'f1', 'g1', 'h3', 'i1', 'j1', 'k1', 'l1']
```

data_from_coefficients(*ainvs*)

Return elliptic curve data for the curve with given Weierstrass coefficients.

EXAMPLES:

```
sage: d = CremonaDatabase().data_from_coefficients([1, -1, 1, 31, 128])
sage: d['conductor']
1953
sage: d['cremona_label']
'1953c1'
sage: d['rank']
1
sage: d['torsion_order']
2
```

```
>>> from sage.all import *
>>> d = CremonaDatabase().data_from_coefficients([Integer(1), -Integer(1),␣
↪Integer(1), Integer(31), Integer(128)])
>>> d['conductor']
1953
>>> d['cremona_label']
'1953c1'
>>> d['rank']
1
>>> d['torsion_order']
2
```

Check that Issue #17904 is fixed:

```
sage: ai = EllipticCurve('100467a2').ainvs() # optional - database_cremona_
↪ellcurve
sage: 'gens' in CremonaDatabase().data_from_coefficients(ai) # optional -␣
↪database_cremona_ellcurve
True
```

```
>>> from sage.all import *
>>> ai = EllipticCurve('100467a2').ainvs() # optional - database_cremona_
↪ellcurve
>>> 'gens' in CremonaDatabase().data_from_coefficients(ai) # optional -␣
↪database_cremona_ellcurve
True
```

elliptic_curve(*label*)

Return an elliptic curve with given label with some data about it from the database pre-filled in.

INPUT:

- `label` – string (Cremona or LMFDB label)

OUTPUT: an `sage.schemes.elliptic_curves.ell_rational_field.EllipticCurve_rational_field`

Note

For more details on LMFDB labels see *parse_lmfdb_label()*.

EXAMPLES:

```
sage: c = CremonaDatabase()
sage: c.elliptic_curve('11a1')
Elliptic Curve defined by y^2 + y = x^3 - x^2 - 10*x - 20 over Rational Field
sage: c.elliptic_curve('12001a1')    # optional - database_cremona_ellcurve
Elliptic Curve defined by y^2 + x*y = x^3 - 101*x + 382 over Rational Field
sage: c.elliptic_curve('48c1')
Traceback (most recent call last):
...
ValueError: There is no elliptic curve with label 48c1 in the database
```

```
>>> from sage.all import *
>>> c = CremonaDatabase()
>>> c.elliptic_curve('11a1')
Elliptic Curve defined by y^2 + y = x^3 - x^2 - 10*x - 20 over Rational Field
>>> c.elliptic_curve('12001a1')    # optional - database_cremona_ellcurve
Elliptic Curve defined by y^2 + x*y = x^3 - 101*x + 382 over Rational Field
>>> c.elliptic_curve('48c1')
Traceback (most recent call last):
...
ValueError: There is no elliptic curve with label 48c1 in the database
```

You can also use LMFDB labels:

```
sage: c.elliptic_curve('462.f3')
Elliptic Curve defined by y^2 + x*y = x^3 - 363*x + 1305 over Rational Field
```

```
>>> from sage.all import *
>>> c.elliptic_curve('462.f3')
Elliptic Curve defined by y^2 + x*y = x^3 - 363*x + 1305 over Rational Field
```

elliptic_curve_from_ainvs(*ainvs*)

Return the elliptic curve in the database of with minimal `ainvs` if it exists.

This raises a `RuntimeError` exception otherwise.

INPUT:

- `ainvs` – list (5-tuple of int's); the minimal Weierstrass model for an elliptic curve

OUTPUT: EllipticCurve

EXAMPLES:

```
sage: c = CremonaDatabase()
sage: c.elliptic_curve_from_ainvs([0, -1, 1, -10, -20])
Elliptic Curve defined by y^2 + y = x^3 - x^2 - 10*x - 20 over Rational Field
sage: c.elliptic_curve_from_ainvs([1, 0, 0, -101, 382])  # optional -␣
↪database_cremona_ellcurve
Elliptic Curve defined by y^2 + x*y = x^3 - 101*x + 382 over Rational Field
```

```
>>> from sage.all import *
>>> c = CremonaDatabase()
>>> c.elliptic_curve_from_ainvs([Integer(0), -Integer(1), Integer(1), -
↪Integer(10), -Integer(20)])
```

(continues on next page)

(continued from previous page)

```
Elliptic Curve defined by y^2 + y = x^3 - x^2 - 10*x - 20 over Rational Field
>>> c.elliptic_curve_from_ainvs([Integer(1), Integer(0), Integer(0), -
↪Integer(101), Integer(382)])  # optional - database_cremona_ellcurve
Elliptic Curve defined by y^2 + x*y = x^3 - 101*x + 382 over Rational Field
```

Old (pre-2006) Cremona labels are also allowed:

```
sage: c.elliptic_curve('9450KKKK1')
Elliptic Curve defined by y^2 + x*y + y = x^3 - x^2 - 5*x + 7 over Rational␣
↪Field
```

```
>>> from sage.all import *
>>> c.elliptic_curve('9450KKKK1')
Elliptic Curve defined by y^2 + x*y + y = x^3 - x^2 - 5*x + 7 over Rational␣
↪Field
```

Make sure Issue #12565 is fixed:

```
sage: c.elliptic_curve('10a1')
Traceback (most recent call last):
...
ValueError: There is no elliptic curve with label 10a1 in the database
```

```
>>> from sage.all import *
>>> c.elliptic_curve('10a1')
Traceback (most recent call last):
...
ValueError: There is no elliptic curve with label 10a1 in the database
```

isogeny_class(*label*)

Return the isogeny class of elliptic curves that are isogenous to the curve with given Cremona label.

INPUT:

- `label` – string

OUTPUT: list of EllipticCurve objects

EXAMPLES:

```
sage: c = CremonaDatabase()
sage: c.isogeny_class('11a1')
[Elliptic Curve defined by y^2 + y = x^3 - x^2 - 10*x - 20 over Rational␣
↪Field,
 Elliptic Curve defined by y^2 + y = x^3 - x^2 - 7820*x - 263580 over␣
↪Rational Field,
 Elliptic Curve defined by y^2 + y = x^3 - x^2 over Rational Field]
sage: c.isogeny_class('12001a1')   # optional - database_cremona_ellcurve
[Elliptic Curve defined by y^2 + x*y = x^3 - 101*x + 382 over Rational Field]
```

```
>>> from sage.all import *
>>> c = CremonaDatabase()
>>> c.isogeny_class('11a1')
```

(continues on next page)

(continued from previous page)

```
[Elliptic Curve defined by y^2 + y = x^3 - x^2 - 10*x - 20 over Rational␣
↪Field,
 Elliptic Curve defined by y^2 + y = x^3 - x^2 - 7820*x - 263580 over␣
↪Rational Field,
 Elliptic Curve defined by y^2 + y = x^3 - x^2 over Rational Field]
>>> c.isogeny_class('12001a1')   # optional - database_cremona_ellcurve
[Elliptic Curve defined by y^2 + x*y = x^3 - 101*x + 382 over Rational Field]
```

isogeny_classes(*conductor*)

Return the allcurves data (ainvariants, rank and torsion) for the elliptic curves in the database of given conductor as a list of lists, one for each isogeny class. The curve with number 1 is always listed first.

EXAMPLES:

```
sage: c = CremonaDatabase()
sage: c.isogeny_classes(11)
[[[0, -1, 1, -10, -20], 0, 5],
 [[0, -1, 1, -7820, -263580], 0, 1],
 [[0, -1, 1, 0, 0], 0, 5]]]
sage: c.isogeny_classes(12001)   # optional - database_cremona_ellcurve
[[[1, 0, 0, -101, 382], 1, 1]],
 [[[0, 0, 1, -247, 1494], 1, 1]],
 [[[0, 0, 1, -4, -18], 1, 1]],
 [[[0, 1, 1, -10, 18], 1, 1]]]
```

```
>>> from sage.all import *
>>> c = CremonaDatabase()
>>> c.isogeny_classes(Integer(11))
[[[0, -1, 1, -10, -20], 0, 5],
 [[0, -1, 1, -7820, -263580], 0, 1],
 [[0, -1, 1, 0, 0], 0, 5]]]
>>> c.isogeny_classes(Integer(12001))   # optional - database_cremona_ellcurve
[[[1, 0, 0, -101, 382], 1, 1]],
 [[[0, 0, 1, -247, 1494], 1, 1]],
 [[[0, 0, 1, -4, -18], 1, 1]],
 [[[0, 1, 1, -10, 18], 1, 1]]]
```

iter(*conductors*)

Return an iterator through all curves in the database with given conductors.

INPUT:

- `conductors` – list or generator of ints

OUTPUT: generator that iterates over EllipticCurve objects

EXAMPLES:

```
sage: [e.cremona_label() for e in CremonaDatabase().iter([11..15])]
['11a1', '11a2', '11a3', '14a1', '14a2', '14a3', '14a4', '14a5',
 '14a6', '15a1', '15a2', '15a3', '15a4', '15a5', '15a6', '15a7', '15a8']
```

```
>>> from sage.all import *
>>> [e.cremona_label() for e in CremonaDatabase().iter((ellipsis_
```

(continues on next page)

(continued from previous page)

```
↪range(Integer(11),Ellipsis,Integer(15)))))]
['11a1', '11a2', '11a3', '14a1', '14a2', '14a3', '14a4', '14a5',
 '14a6', '15a1', '15a2', '15a3', '15a4', '15a5', '15a6', '15a7', '15a8']
```

iter_optimal(*conductors*)

Return an iterator through all optimal curves in the database with given conductors.

INPUT:

- `conductors` – list or generator of ints

OUTPUT: generator that iterates over EllipticCurve objects

EXAMPLES:

We list optimal curves with conductor up to 20:

```
sage: [e.cremona_label() for e in CremonaDatabase().iter_optimal([11..20])]
['11a1', '14a1', '15a1', '17a1', '19a1', '20a1']
```

```
>>> from sage.all import *
>>> [e.cremona_label() for e in CremonaDatabase().iter_optimal((ellipsis_
↪range(Integer(11),Ellipsis,Integer(20))))]
['11a1', '14a1', '15a1', '17a1', '19a1', '20a1']
```

Note the unfortunate 990h3 special case:

```
sage: [e.cremona_label() for e in CremonaDatabase().iter_optimal([990])]
['990a1', '990b1', '990c1', '990d1', '990e1', '990f1', '990g1', '990h3',
↪'990i1', '990j1', '990k1', '990l1']
```

```
>>> from sage.all import *
>>> [e.cremona_label() for e in CremonaDatabase().iter_
↪optimal([Integer(990)])]
['990a1', '990b1', '990c1', '990d1', '990e1', '990f1', '990g1', '990h3',
↪'990i1', '990j1', '990k1', '990l1']
```

largest_conductor()

The largest conductor for which the database is complete.

OUTPUT: integer; largest conductor

EXAMPLES:

```
sage: c = CremonaDatabase('cremona mini')
sage: c.largest_conductor()
9999
```

```
>>> from sage.all import *
>>> c = CremonaDatabase('cremona mini')
>>> c.largest_conductor()
9999
```

list(*conductors*)

Return a list of all curves with given conductors.

INPUT:

- `conductors` – list or generator of ints

OUTPUT: list of EllipticCurve objects

EXAMPLES:

```
sage: CremonaDatabase().list([37])
[Elliptic Curve defined by y^2 + y = x^3 - x over Rational Field,
 Elliptic Curve defined by y^2 + y = x^3 + x^2 - 23*x - 50 over Rational␣
↪Field,
 Elliptic Curve defined by y^2 + y = x^3 + x^2 - 1873*x - 31833 over Rational␣
↪Field,
 Elliptic Curve defined by y^2 + y = x^3 + x^2 - 3*x + 1 over Rational Field]
```

```
>>> from sage.all import *
>>> CremonaDatabase().list([Integer(37)])
[Elliptic Curve defined by y^2 + y = x^3 - x over Rational Field,
 Elliptic Curve defined by y^2 + y = x^3 + x^2 - 23*x - 50 over Rational␣
↪Field,
 Elliptic Curve defined by y^2 + y = x^3 + x^2 - 1873*x - 31833 over Rational␣
↪Field,
 Elliptic Curve defined by y^2 + y = x^3 + x^2 - 3*x + 1 over Rational Field]
```

list_optimal (*conductors*)

Return a list of all optimal curves with given conductors.

INPUT:

- `conductors` – list or generator of ints list of EllipticCurve objects

OUTPUT: list of EllipticCurve objects

EXAMPLES:

```
sage: CremonaDatabase().list_optimal([37])
[Elliptic Curve defined by y^2 + y = x^3 - x over Rational Field,
 Elliptic Curve defined by y^2 + y = x^3 + x^2 - 23*x - 50 over Rational␣
↪Field]
```

```
>>> from sage.all import *
>>> CremonaDatabase().list_optimal([Integer(37)])
[Elliptic Curve defined by y^2 + y = x^3 - x over Rational Field,
 Elliptic Curve defined by y^2 + y = x^3 + x^2 - 23*x - 50 over Rational␣
↪Field]
```

number_of_curves (*N=0*, *i=0*)

Return the number of curves stored in the database with conductor N. If $N = 0$, returns the total number of curves in the database.

If i is nonzero, returns the number of curves in the i-th isogeny class. If i is a Cremona letter code, e.g., 'a' or 'bc', it is converted to the corresponding number.

INPUT:

- `N` – integer
- `i` – integer or string

OUTPUT: integer

EXAMPLES:

```
sage: c = CremonaDatabase()
sage: c.number_of_curves(11)
3
sage: c.number_of_curves(37)
4
sage: c.number_of_curves(990)
42
sage: num = c.number_of_curves()
```

```
>>> from sage.all import *
>>> c = CremonaDatabase()
>>> c.number_of_curves(Integer(11))
3
>>> c.number_of_curves(Integer(37))
4
>>> c.number_of_curves(Integer(990))
42
>>> num = c.number_of_curves()
```

number_of_isogeny_classes(*N=0*)

Return the number of isogeny classes of curves in the database of conductor N. If N is 0, return the total number of isogeny classes of curves in the database.

INPUT:

- N – integer

OUTPUT: integer

EXAMPLES:

```
sage: c = CremonaDatabase()
sage: c.number_of_isogeny_classes(11)
1
sage: c.number_of_isogeny_classes(37)
2
sage: num = c.number_of_isogeny_classes()
```

```
>>> from sage.all import *
>>> c = CremonaDatabase()
>>> c.number_of_isogeny_classes(Integer(11))
1
>>> c.number_of_isogeny_classes(Integer(37))
2
>>> num = c.number_of_isogeny_classes()
```

random()

Return a random curve from the database.

EXAMPLES:

```
sage: CremonaDatabase().random() # random -- depends on database installed
Elliptic Curve defined by y^2 + x*y  = x^3 - x^2 - 224*x + 3072 over Rational␣
↪Field
```

```
>>> from sage.all import *
>>> CremonaDatabase().random() # random -- depends on database installed
Elliptic Curve defined by y^2 + x*y  = x^3 - x^2 - 224*x + 3072 over Rational␣
↪Field
```

smallest_conductor()

The smallest conductor for which the database is complete: always 1.

OUTPUT: integer; smallest conductor

Note

This always returns the integer 1, since that is the smallest conductor for which the database is complete, although there are no elliptic curves of conductor 1. The smallest conductor of a curve in the database is 11.

EXAMPLES:

```
sage: CremonaDatabase().smallest_conductor()
1
```

```
>>> from sage.all import *
>>> CremonaDatabase().smallest_conductor()
1
```

sage.databases.cremona.**build**(*name*, *data_tgz*, *largest_conductor=0*, *mini=False*, *decompress=True*)

Build the CremonaDatabase with given name from scratch using the data_tgz tarball.

Note

For data up to level 350000, this function takes about 3m40s. The resulting database occupies 426MB disk space.

To create the large Cremona database from Cremona's data_tgz tarball, obtainable from http://homepages.warwick.ac.uk/staff/J.E.Cremona/ftp/data/, run the following command:

```
sage: d = sage.databases.cremona.build('cremona','ecdata.tgz')   # not tested
```

```
>>> from sage.all import *
>>> d = sage.databases.cremona.build('cremona','ecdata.tgz')   # not tested
```

sage.databases.cremona.**class_to_int**(*k*)

Convert class id string into an integer.

Note that this is the inverse of *cremona_letter_code()*.

EXAMPLES:

```
sage: import sage.databases.cremona as cremona
sage: cremona.class_to_int('ba')
26
sage: cremona.class_to_int('cremona')
821863562
sage: cremona.cremona_letter_code(821863562)
'cremona'
```

```
>>> from sage.all import *
>>> import sage.databases.cremona as cremona
>>> cremona.class_to_int('ba')
26
>>> cremona.class_to_int('cremona')
821863562
>>> cremona.cremona_letter_code(Integer(821863562))
'cremona'
```

sage.databases.cremona.**cremona_letter_code**(*n*)

Return the Cremona letter code corresponding to an integer.

For example, 0 - a 25 - z 26 - ba 51 - bz 52 - ca 53 - cb etc.

Note

This is just the base 26 representation of n, where a=0, b=1, …, z=25. This extends the old Cremona notation (counting from 0) for the first 26 classes, and is different for classes above 26.

INPUT:

- n – nonnegative integer

OUTPUT: string

EXAMPLES:

```
sage: from sage.databases.cremona import cremona_letter_code
sage: cremona_letter_code(0)
'a'
sage: cremona_letter_code(26)
'ba'
sage: cremona_letter_code(27)
'bb'
sage: cremona_letter_code(521)
'ub'
sage: cremona_letter_code(53)
'cb'
sage: cremona_letter_code(2005)
'czd'
```

```
>>> from sage.all import *
>>> from sage.databases.cremona import cremona_letter_code
>>> cremona_letter_code(Integer(0))
'a'
```

(continues on next page)

(continued from previous page)

```
>>> cremona_letter_code(Integer(26))
'ba'
>>> cremona_letter_code(Integer(27))
'bb'
>>> cremona_letter_code(Integer(521))
'ub'
>>> cremona_letter_code(Integer(53))
'cb'
>>> cremona_letter_code(Integer(2005))
'czd'
```

sage.databases.cremona.**cremona_to_lmfdb**(*cremona_label*, *CDB=None*)

Convert a Cremona label into an LMFDB label.

See *parse_lmfdb_label()* for an explanation of LMFDB labels.

INPUT:

- cremona_label – string, the Cremona label of a curve; this can be the label of a curve (e.g. '990j1') or of an isogeny class (e.g. '990j')
- CDB – the Cremona database in which to look up the isogeny classes of the same conductor

OUTPUT: lmfdb_label; string, the corresponding LMFDB label

EXAMPLES:

```
sage: from sage.databases.cremona import cremona_to_lmfdb, lmfdb_to_cremona
sage: cremona_to_lmfdb('990j1')
'990.h3'
sage: lmfdb_to_cremona('990.h3')
'990j1'
```

```
>>> from sage.all import *
>>> from sage.databases.cremona import cremona_to_lmfdb, lmfdb_to_cremona
>>> cremona_to_lmfdb('990j1')
'990.h3'
>>> lmfdb_to_cremona('990.h3')
'990j1'
```

sage.databases.cremona.**is_optimal_id**(*id*)

Return True if the Cremona id refers to an optimal curve, and False otherwise.

The curve is optimal if the id, which is of the form [letter code][number] has number 1.

Note

990h3 is the optimal curve in that class, so doesn't obey this rule.

INPUT:

- id – string of form letter code followed by an integer, e.g., a3, bb5, etc.

OUTPUT: boolean

EXAMPLES:

```
sage: from sage.databases.cremona import is_optimal_id
sage: is_optimal_id('b1')
True
sage: is_optimal_id('bb1')
True
sage: is_optimal_id('c1')
True
sage: is_optimal_id('c2')
False
```

```
>>> from sage.all import *
>>> from sage.databases.cremona import is_optimal_id
>>> is_optimal_id('b1')
True
>>> is_optimal_id('bb1')
True
>>> is_optimal_id('c1')
True
>>> is_optimal_id('c2')
False
```

sage.databases.cremona.**lmfdb_to_cremona**(*lmfdb_label*, *CDB=None*)

Convert an LMFDB label into a Cremona label.

See *parse_lmfdb_label()* for an explanation of LMFDB labels.

INPUT:

- `lmfdb_label` – string, the LMFDB label of a curve; this can be the label of a curve (e.g. '990.j1') or of an isogeny class (e.g. '990.j')
- `CDB` – the Cremona database in which to look up the isogeny classes of the same conductor

OUTPUT: `cremona_label`; a string, the corresponding Cremona label

EXAMPLES:

```
sage: from sage.databases.cremona import cremona_to_lmfdb, lmfdb_to_cremona
sage: lmfdb_to_cremona('990.h3')
'990j1'
sage: cremona_to_lmfdb('990j1')
'990.h3'
```

```
>>> from sage.all import *
>>> from sage.databases.cremona import cremona_to_lmfdb, lmfdb_to_cremona
>>> lmfdb_to_cremona('990.h3')
'990j1'
>>> cremona_to_lmfdb('990j1')
'990.h3'
```

sage.databases.cremona.**old_cremona_letter_code**(*n*)

Return the *old* Cremona letter code corresponding to an integer.

For example:

```
1  --> A
26 --> Z
27 --> AA
52 --> ZZ
53 --> AAA
etc.
```

INPUT:

- `n` – integer

OUTPUT: string

EXAMPLES:

```
sage: from sage.databases.cremona import old_cremona_letter_code
sage: old_cremona_letter_code(1)
'A'
sage: old_cremona_letter_code(26)
'Z'
sage: old_cremona_letter_code(27)
'AA'
sage: old_cremona_letter_code(521)
'AAAAAAAAAAAAAAAAAAAAA'
sage: old_cremona_letter_code(53)
'AAA'
sage: old_cremona_letter_code(2005)
'CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC'
```

```
>>> from sage.all import *
>>> from sage.databases.cremona import old_cremona_letter_code
>>> old_cremona_letter_code(Integer(1))
'A'
>>> old_cremona_letter_code(Integer(26))
'Z'
>>> old_cremona_letter_code(Integer(27))
'AA'
>>> old_cremona_letter_code(Integer(521))
'AAAAAAAAAAAAAAAAAAAAA'
>>> old_cremona_letter_code(Integer(53))
'AAA'
>>> old_cremona_letter_code(Integer(2005))
'CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC'
```

`sage.databases.cremona.`**`parse_cremona_label`**(*label*, *numerical_class_code=False*)

Given a Cremona label that defines an elliptic curve, e.g., 11a1 or 37b3, parse the label and return the conductor, isogeny class label, and number.

For this function, the curve number may be omitted, in which case it defaults to 1. If the curve number and isogeny class are both omitted (label is just a string representing a conductor), then the isogeny class defaults to 'a' and the number to 1. Valid labels consist of one or more digits, followed by zero or more letters (either all in upper case for an old Cremona label, or all in lower case), followed by zero or more digits.

INPUT:

- `label` – string; a valid Cremona elliptic curve label

- `numerical_class_code` – boolean (default: `False`); if `True`, convert the isogeny class label from a letter code in base 26 to an integer. This is useful for sorting.

OUTPUT:

- integer; the conductor
- string or integer; the isogeny class label
- integer; the number

EXAMPLES:

```
sage: from sage.databases.cremona import parse_cremona_label
sage: parse_cremona_label('37a2')
(37, 'a', 2)
sage: parse_cremona_label('37b1')
(37, 'b', 1)
sage: parse_cremona_label('10bb2')
(10, 'bb', 2)
sage: parse_cremona_label('11a')
(11, 'a', 1)
sage: parse_cremona_label('11')
(11, 'a', 1)
```

```
>>> from sage.all import *
>>> from sage.databases.cremona import parse_cremona_label
>>> parse_cremona_label('37a2')
(37, 'a', 2)
>>> parse_cremona_label('37b1')
(37, 'b', 1)
>>> parse_cremona_label('10bb2')
(10, 'bb', 2)
>>> parse_cremona_label('11a')
(11, 'a', 1)
>>> parse_cremona_label('11')
(11, 'a', 1)
```

Valid old Cremona labels are allowed:

```
sage: parse_cremona_label('17CCCC')
(17, 'dc', 1)
sage: parse_cremona_label('5AB2')
Traceback (most recent call last):
...
ValueError: 5AB2 is not a valid Cremona label
```

```
>>> from sage.all import *
>>> parse_cremona_label('17CCCC')
(17, 'dc', 1)
>>> parse_cremona_label('5AB2')
Traceback (most recent call last):
...
ValueError: 5AB2 is not a valid Cremona label
```

When `numerical_class_code` is `True`, the output is a triple of integers:

```
sage: from sage.databases.cremona import parse_cremona_label
sage: parse_cremona_label('100800hj2')
(100800, 'hj', 2)
sage: parse_cremona_label('100800hj2', numerical_class_code=True)
(100800, 191, 2)
```

```
>>> from sage.all import *
>>> from sage.databases.cremona import parse_cremona_label
>>> parse_cremona_label('100800hj2')
(100800, 'hj', 2)
>>> parse_cremona_label('100800hj2', numerical_class_code=True)
(100800, 191, 2)
```

sage.databases.cremona.**parse_lmfdb_label**(*label*, *numerical_class_code=False*)

Given an LMFDB label that defines an elliptic curve, e.g., 11.a1 or 37.b3, parse the label and return the conductor, isogeny class label, and number.

The LMFDB label (named after the L-functions and modular forms database), is determined by the following two orders:

- Isogeny classes with the same conductor are ordered lexicographically by the coefficients in the q-expansion of the associated modular form.
- Curves within the same isogeny class are ordered lexicographically by the a-invariants of the minimal model.

The format is <conductor>.<iso><curve>, where the isogeny class is encoded using the same base-26 encoding into letters used in Cremona's labels. For example, 990.h3 is the same as Cremona's 990j1

For this function, the curve number may be omitted, in which case it defaults to 1. If the curve number and isogeny class are both omitted (label is just a string representing a conductor), then the isogeny class defaults to 'a' and the number to 1.

INPUT:

- `label` – str
- `numerical_class_code` – boolean (default: `False`); if `True`, convert the isogeny class label from a letter code in base 26 to an integer. This is useful for sorting.

OUTPUT:

- `int` – the conductor
- `str` or `int` – the isogeny class label
- `int` – the number

EXAMPLES:

```
sage: from sage.databases.cremona import parse_lmfdb_label
sage: parse_lmfdb_label('37.a2')
(37, 'a', 2)
sage: parse_lmfdb_label('37.b')
(37, 'b', 1)
sage: parse_lmfdb_label('10.bb2')
(10, 'bb', 2)
```

```
>>> from sage.all import *
>>> from sage.databases.cremona import parse_lmfdb_label
>>> parse_lmfdb_label('37.a2')
(37, 'a', 2)
>>> parse_lmfdb_label('37.b')
(37, 'b', 1)
>>> parse_lmfdb_label('10.bb2')
(10, 'bb', 2)
```

When `numerical_class_code` is `True`, the output is a triple of integers:

```
sage: from sage.databases.cremona import parse_lmfdb_label
sage: parse_lmfdb_label('100800.bg4')
(100800, 'bg', 4)
sage: parse_lmfdb_label('100800.bg4', numerical_class_code=True)
(100800, 32, 4)
```

```
>>> from sage.all import *
>>> from sage.databases.cremona import parse_lmfdb_label
>>> parse_lmfdb_label('100800.bg4')
(100800, 'bg', 4)
>>> parse_lmfdb_label('100800.bg4', numerical_class_code=True)
(100800, 32, 4)
```

sage.databases.cremona.**sort_key**(*key1*)

Comparison key for curve id strings.

Note

Not the same as standard lexicographic order!

EXAMPLES:

```
sage: from sage.databases.cremona import sort_key
sage: l = ['ba1', 'z1']
sage: sorted(l, key=sort_key)
['z1', 'ba1']
```

```
>>> from sage.all import *
>>> from sage.databases.cremona import sort_key
>>> l = ['ba1', 'z1']
>>> sorted(l, key=sort_key)
['z1', 'ba1']
```

sage.databases.cremona.**split_code**(*key*)

Split class + curve id string into its two parts.

EXAMPLES:

```
sage: import sage.databases.cremona as cremona
sage: cremona.split_code('ba2')
('ba', '2')
```

(continues on next page)

(continued from previous page)

```
sage: cremona.split_code('42')
Traceback (most recent call last):
...
ValueError: invalid curve ID: '42'
```

```
>>> from sage.all import *
>>> import sage.databases.cremona as cremona
>>> cremona.split_code('ba2')
('ba', '2')
>>> cremona.split_code('42')
Traceback (most recent call last):
...
ValueError: invalid curve ID: '42'
```

2.6 The Stein-Watkins table of elliptic curves

Sage gives access to the Stein-Watkins table of elliptic curves, via the optional database_stein_watkins package that you must install. This is a huge database of elliptic curves. You can install the database (a 2.6GB package) with the command

```
sage -i database_stein_watkins
```

You can also automatically download a small version, which takes much less time, via the optional database_stein_watkins_mini package using the command

```
sage -i database_stein_watkins_mini
```

This database covers a wide range of conductors, but unlike the *Cremona database*, this database need not list all curves of a given conductor. It lists the curves whose coefficients are not "too large" (see [SW2002]).

- The command `SteinWatkinsAllData(n)` returns an iterator over the curves in the n-th Stein-Watkins table, which contains elliptic curves of conductor between $n10^5$ and $(n+1)10^5$. Here n can be between 0 and 999, inclusive.
- The command `SteinWatkinsPrimeData(n)` returns an iterator over the curves in the n-th Stein-Watkins prime table, which contains prime conductor elliptic curves of conductor between $n10^8$ and $(n+1)10^8$. Here n varies between 0 and 99, inclusive.

EXAMPLES: We obtain the first table of elliptic curves.

```
sage: d = SteinWatkinsAllData(0)
sage: d
Stein-Watkins Database a.0 Iterator
```

```
>>> from sage.all import *
>>> d = SteinWatkinsAllData(Integer(0))
>>> d
Stein-Watkins Database a.0 Iterator
```

We type `next(d)` to get each isogeny class of curves from `d`:

```
sage: # optional - database_stein_watkins
sage: C = next(d)
```

(continues on next page)

(continued from previous page)

```
sage: C
Stein-Watkins isogeny class of conductor 11
sage: next(d)
Stein-Watkins isogeny class of conductor 14
sage: next(d)
Stein-Watkins isogeny class of conductor 15
```

```
>>> from sage.all import *
>>> # optional - database_stein_watkins
>>> C = next(d)
>>> C
Stein-Watkins isogeny class of conductor 11
>>> next(d)
Stein-Watkins isogeny class of conductor 14
>>> next(d)
Stein-Watkins isogeny class of conductor 15
```

An isogeny class has a number of attributes that give data about the isogeny class, such as the rank, equations of curves, conductor, leading coefficient of L-function, etc.

```
sage: # optional - database_stein_watkins
sage: C.data
['11', '[11]', '0', '0.253842', '25', '+*1']
sage: C.curves
[[[0, -1, 1, 0, 0], '(1)', '1', '5'],
 [[0, -1, 1, -10, -20], '(5)', '1', '5'],
 [[0, -1, 1, -7820, -263580], '(1)', '1', '1']]
sage: C.conductor
11
sage: C.leading_coefficient
'0.253842'
sage: C.modular_degree
'+*1'
sage: C.rank
0
sage: C.isogeny_number
'25'
```

```
>>> from sage.all import *
>>> # optional - database_stein_watkins
>>> C.data
['11', '[11]', '0', '0.253842', '25', '+*1']
>>> C.curves
[[[0, -1, 1, 0, 0], '(1)', '1', '5'],
 [[0, -1, 1, -10, -20], '(5)', '1', '5'],
 [[0, -1, 1, -7820, -263580], '(1)', '1', '1']]
>>> C.conductor
11
>>> C.leading_coefficient
'0.253842'
>>> C.modular_degree
'+*1'
```

(continues on next page)

(continued from previous page)

```
>>> C.rank
0
>>> C.isogeny_number
'25'
```

If we were to continue typing `next(d)` we would iterate over all curves in the Stein-Watkins database up to conductor 10^5. We could also type `for C in d: ...`

To access the data file starting at 10^5 do the following:

```
sage: d = SteinWatkinsAllData(1)
sage: C = next(d)                                    # optional - database_stein_watkins
sage: C                                              # optional - database_stein_watkins
Stein-Watkins isogeny class of conductor 100002
sage: C.curves                                       # optional - database_stein_watkins
[[[1, 1, 0, 112, 0], '(8,1,2,1)', 'X', '2'],
 [[1, 1, 0, -448, -560], '[4,2,1,2]', 'X', '2']]
```

```
>>> from sage.all import *
>>> d = SteinWatkinsAllData(Integer(1))
>>> C = next(d)                                      # optional - database_stein_watkins
>>> C                                                # optional - database_stein_watkins
Stein-Watkins isogeny class of conductor 100002
>>> C.curves                                         # optional - database_stein_watkins
[[[1, 1, 0, 112, 0], '(8,1,2,1)', 'X', '2'],
 [[1, 1, 0, -448, -560], '[4,2,1,2]', 'X', '2']]
```

Next we access the prime-conductor data:

```
sage: d = SteinWatkinsPrimeData(0)
sage: C = next(d)                                    # optional - database_stein_watkins
sage: C                                              # optional - database_stein_watkins
Stein-Watkins isogeny class of conductor 11
```

```
>>> from sage.all import *
>>> d = SteinWatkinsPrimeData(Integer(0))
>>> C = next(d)                                      # optional - database_stein_watkins
>>> C                                                # optional - database_stein_watkins
Stein-Watkins isogeny class of conductor 11
```

Each call `next(d)` gives another elliptic curve of prime conductor:

```
sage: # optional - database_stein_watkins
sage: C = next(d)
sage: C
Stein-Watkins isogeny class of conductor 17
sage: C.curves
[[[1, -1, 1, -1, 0], '[1]', '1', '4'],
 [[1, -1, 1, -6, -4], '[2]', '1', '2x'],
 [[1, -1, 1, -1, -14], '(4)', '1', '4'],
 [[1, -1, 1, -91, -310], '[1]', '1', '2']]
sage: C = next(d)
```

(continues on next page)

(continued from previous page)

```
sage: C
Stein-Watkins isogeny class of conductor 19
```

```
>>> from sage.all import *
>>> # optional - database_stein_watkins
>>> C = next(d)
>>> C
Stein-Watkins isogeny class of conductor 17
>>> C.curves
[[[1, -1, 1, -1, 0], '[1]', '1', '4'],
 [[1, -1, 1, -6, -4], '[2]', '1', '2x'],
 [[1, -1, 1, -1, -14], '(4)', '1', '4'],
 [[1, -1, 1, -91, -310], '[1]', '1', '2']]
>>> C = next(d)
>>> C
Stein-Watkins isogeny class of conductor 19
```

REFERENCE:

- [SW2002]

class sage.databases.stein_watkins.**SteinWatkinsAllData**(*num*)

Bases: object

Class for iterating through one of the Stein-Watkins database files for all conductors.

iter_levels()

Iterate through the curve classes, but grouped into lists by level.

EXAMPLES:

```
sage: d = SteinWatkinsAllData(1)
sage: E = d.iter_levels()
sage: next(E)                                # optional - database_stein_watkins
[Stein-Watkins isogeny class of conductor 100002]
sage: next(E)                                # optional - database_stein_watkins
[Stein-Watkins isogeny class of conductor 100005,
Stein-Watkins isogeny class of conductor 100005]
sage: next(E)                                # optional - database_stein_watkins
[Stein-Watkins isogeny class of conductor 100007]
```

```
>>> from sage.all import *
>>> d = SteinWatkinsAllData(Integer(1))
>>> E = d.iter_levels()
>>> next(E)                                  # optional - database_stein_watkins
[Stein-Watkins isogeny class of conductor 100002]
>>> next(E)                                  # optional - database_stein_watkins
[Stein-Watkins isogeny class of conductor 100005,
Stein-Watkins isogeny class of conductor 100005]
>>> next(E)                                  # optional - database_stein_watkins
[Stein-Watkins isogeny class of conductor 100007]
```

next()

class sage.databases.stein_watkins.**SteinWatkinsIsogenyClass**(*conductor*)

Bases: object

class sage.databases.stein_watkins.**SteinWatkinsPrimeData**(*num*)

Bases: *SteinWatkinsAllData*

sage.databases.stein_watkins.**ecdb_num_curves**(*max_level=200000*)

Return a list whose N-th entry, for `0 <= N <= max_level`, is the number of elliptic curves of conductor N in the database.

EXAMPLES:

```
sage: sage.databases.stein_watkins.ecdb_num_curves(100) # optional - database_
↪stein_watkins
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 3, 0, 0, 6, 8, 0, 4, 0, 3, 4, 6, 0, 0,
 6, 0, 5, 4, 0, 0, 8, 0, 4, 4, 4, 3, 4, 4, 5, 4, 4, 0, 6, 1, 2, 8, 2, 0,
 6, 4, 8, 2, 2, 1, 6, 4, 6, 7, 3, 0, 0, 1, 4, 6, 4, 2, 12, 1, 0, 2, 4, 0,
 6, 2, 0, 12, 1, 6, 4, 1, 8, 0, 2, 1, 6, 2, 0, 0, 1, 3, 16, 4, 3, 0, 2,
 0, 8, 0, 6, 11, 4]
```

```
>>> from sage.all import *
>>> sage.databases.stein_watkins.ecdb_num_curves(Integer(100)) # optional -␣
↪database_stein_watkins
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 3, 0, 0, 6, 8, 0, 4, 0, 3, 4, 6, 0, 0,
 6, 0, 5, 4, 0, 0, 8, 0, 4, 4, 4, 3, 4, 4, 5, 4, 4, 0, 6, 1, 2, 8, 2, 0,
 6, 4, 8, 2, 2, 1, 6, 4, 6, 7, 3, 0, 0, 1, 4, 6, 4, 2, 12, 1, 0, 2, 4, 0,
 6, 2, 0, 12, 1, 6, 4, 1, 8, 0, 2, 1, 6, 2, 0, 0, 1, 3, 16, 4, 3, 0, 2,
 0, 8, 0, 6, 11, 4]
```

2.7 John Jones's tables of number fields

In order to use the Jones database, the optional database_jones_numfield package must be installed using the Sage command

```
sage -i database_jones_numfield
```

This is a table of number fields with bounded ramification and degree ≤ 6. You can query the database for all number fields in Jones's tables with bounded ramification and degree.

EXAMPLES: First load the database:

```
sage: J = JonesDatabase()
sage: J
John Jones's table of number fields with bounded ramification and degree <= 6
```

```
>>> from sage.all import *
>>> J = JonesDatabase()
>>> J
John Jones's table of number fields with bounded ramification and degree <= 6
```

List the degree and discriminant of all fields in the database that have ramification at most at 2:

```
sage: [(k.degree(), k.disc()) for k in J.unramified_outside([2])]     # optional -␣
↪database_jones_numfield
[(1, 1), (2, -4), (2, -8), (2, 8), (4, 256), (4, 512), (4, -1024), (4, -2048), (4,␣
↪2048), (4, 2048), (4, 2048)]
```

```
>>> from sage.all import *
>>> [(k.degree(), k.disc()) for k in J.unramified_outside([Integer(2)])]     #␣
↪optional - database_jones_numfield
[(1, 1), (2, -4), (2, -8), (2, 8), (4, 256), (4, 512), (4, -1024), (4, -2048), (4,␣
↪2048), (4, 2048), (4, 2048)]
```

List the discriminants of the fields of degree exactly 2 unramified outside 2:

```
sage: [k.disc() for k in J.unramified_outside([2],2)]                 # optional -␣
↪database_jones_numfield
[-4, -8, 8]
```

```
>>> from sage.all import *
>>> [k.disc() for k in J.unramified_outside([Integer(2)],Integer(2))]
↪# optional - database_jones_numfield
[-4, -8, 8]
```

List the discriminants of cubic field in the database ramified exactly at 3 and 5:

```
sage: [k.disc() for k in J.ramified_at([3,5],3)]                      # optional -␣
↪database_jones_numfield
[-135, -675, -6075, -6075]
sage: factor(6075)
3^5 * 5^2
sage: factor(675)
3^3 * 5^2
sage: factor(135)
3^3 * 5
```

```
>>> from sage.all import *
>>> [k.disc() for k in J.ramified_at([Integer(3),Integer(5)],Integer(3))]           ␣
↪        # optional - database_jones_numfield
[-135, -675, -6075, -6075]
>>> factor(Integer(6075))
3^5 * 5^2
>>> factor(Integer(675))
3^3 * 5^2
>>> factor(Integer(135))
3^3 * 5
```

List all fields in the database ramified at 101:

```
sage: J.ramified_at(101)                                              # optional -␣
↪database_jones_numfield
[Number Field in a with defining polynomial x^2 - 101,
 Number Field in a with defining polynomial x^4 - x^3 + 13*x^2 - 19*x + 361,
 Number Field in a with defining polynomial x^5 + x^4 - 6*x^3 - x^2 + 18*x + 4,
```

(continues on next page)

(continued from previous page)

```
 Number Field in a with defining polynomial x^5 + 2*x^4 + 7*x^3 + 4*x^2 + 11*x - 6,
 Number Field in a with defining polynomial x^5 - x^4 - 40*x^3 - 93*x^2 - 21*x + 17]
```

```
>>> from sage.all import *
>>> J.ramified_at(Integer(101))                                              #␣
↪optional - database_jones_numfield
[Number Field in a with defining polynomial x^2 - 101,
 Number Field in a with defining polynomial x^4 - x^3 + 13*x^2 - 19*x + 361,
 Number Field in a with defining polynomial x^5 + x^4 - 6*x^3 - x^2 + 18*x + 4,
 Number Field in a with defining polynomial x^5 + 2*x^4 + 7*x^3 + 4*x^2 + 11*x - 6,
 Number Field in a with defining polynomial x^5 - x^4 - 40*x^3 - 93*x^2 - 21*x + 17]
```

class sage.databases.jones.**JonesDatabase**

Bases: `object`

get(*S*, *var='a'*)

Return all fields in the database ramified exactly at the primes in S.

INPUT:

- `S` – list or set of primes, or a single prime
- `var` – the name used for the generator of the number fields (default: `'a'`)

EXAMPLES:

```
sage: J = JonesDatabase()               # optional - database_jones_numfield
sage: J.get(163, var='z')               # optional - database_jones_numfield
[Number Field in z with defining polynomial x^2 + 163,
 Number Field in z with defining polynomial x^3 - x^2 - 54*x + 169,
 Number Field in z with defining polynomial x^4 - x^3 - 7*x^2 + 2*x + 9]
sage: J.get([3, 4])                     # optional - database_jones_numfield
Traceback (most recent call last):
...
ValueError: S must be a list of primes
```

```
>>> from sage.all import *
>>> J = JonesDatabase()               # optional - database_jones_numfield
>>> J.get(Integer(163), var='z')              # optional - database_jones_
↪numfield
[Number Field in z with defining polynomial x^2 + 163,
 Number Field in z with defining polynomial x^3 - x^2 - 54*x + 169,
 Number Field in z with defining polynomial x^4 - x^3 - 7*x^2 + 2*x + 9]
>>> J.get([Integer(3), Integer(4)])                    # optional - database_
↪jones_numfield
Traceback (most recent call last):
...
ValueError: S must be a list of primes
```

ramified_at(*S*, *d=None*, *var='a'*)

Return all fields in the database of degree d ramified exactly at the primes in S. The fields are ordered by degree and discriminant.

INPUT:

- `S` – list or set of primes
- `d` – `None` (default, in which case all fields of degree <= 6 are returned) or a positive integer giving the degree of the number fields returned
- `var` – the name used for the generator of the number fields (default: `'a'`)

EXAMPLES:

```
sage: # optional - database_jones_numfield
sage: J = JonesDatabase()
sage: J.ramified_at([101,109])
[]
sage: J.ramified_at([109])
[Number Field in a with defining polynomial x^2 - 109,
 Number Field in a with defining polynomial x^3 - x^2 - 36*x + 4,
 Number Field in a with defining polynomial x^4 - x^3 + 14*x^2 + 34*x + 393]
sage: J.ramified_at(101)
[Number Field in a with defining polynomial x^2 - 101,
 Number Field in a with defining polynomial x^4 - x^3 + 13*x^2 - 19*x + 361,
 Number Field in a with defining polynomial x^5 + x^4 - 6*x^3 - x^2 + 18*x +␣
↪4,
 Number Field in a with defining polynomial x^5 + 2*x^4 + 7*x^3 + 4*x^2 +␣
↪11*x - 6,
 Number Field in a with defining polynomial x^5 - x^4 - 40*x^3 - 93*x^2 -␣
↪21*x + 17]
sage: J.ramified_at((2, 5, 29), 3, 'c')
[Number Field in c with defining polynomial x^3 - x^2 - 8*x - 28,
 Number Field in c with defining polynomial x^3 - x^2 + 10*x + 102,
 Number Field in c with defining polynomial x^3 - x^2 - 48*x - 188,
 Number Field in c with defining polynomial x^3 - x^2 + 97*x - 333]
```

```
>>> from sage.all import *
>>> # optional - database_jones_numfield
>>> J = JonesDatabase()
>>> J.ramified_at([Integer(101),Integer(109)])
[]
>>> J.ramified_at([Integer(109)])
[Number Field in a with defining polynomial x^2 - 109,
 Number Field in a with defining polynomial x^3 - x^2 - 36*x + 4,
 Number Field in a with defining polynomial x^4 - x^3 + 14*x^2 + 34*x + 393]
>>> J.ramified_at(Integer(101))
[Number Field in a with defining polynomial x^2 - 101,
 Number Field in a with defining polynomial x^4 - x^3 + 13*x^2 - 19*x + 361,
 Number Field in a with defining polynomial x^5 + x^4 - 6*x^3 - x^2 + 18*x +␣
↪4,
 Number Field in a with defining polynomial x^5 + 2*x^4 + 7*x^3 + 4*x^2 +␣
↪11*x - 6,
 Number Field in a with defining polynomial x^5 - x^4 - 40*x^3 - 93*x^2 -␣
↪21*x + 17]
>>> J.ramified_at((Integer(2), Integer(5), Integer(29)), Integer(3), 'c')
[Number Field in c with defining polynomial x^3 - x^2 - 8*x - 28,
 Number Field in c with defining polynomial x^3 - x^2 + 10*x + 102,
 Number Field in c with defining polynomial x^3 - x^2 - 48*x - 188,
 Number Field in c with defining polynomial x^3 - x^2 + 97*x - 333]
```

unramified_outside(*S*, *d=None*, *var='a'*)

Return all fields in the database of degree d unramified outside S. If d is omitted, return fields of any degree up to 6. The fields are ordered by degree and discriminant.

INPUT:

- `S` – list or set of primes, or a single prime
- `d` – `None` (default, in which case all fields of degree <= 6 are returned) or a positive integer giving the degree of the number fields returned
- `var` – the name used for the generator of the number fields (default: `'a'`)

EXAMPLES:

```
sage: J = JonesDatabase()              # optional - database_jones_numfield
sage: J.unramified_outside([101,109]) # optional - database_jones_numfield
[Number Field in a with defining polynomial x - 1,
 Number Field in a with defining polynomial x^2 - 101,
 Number Field in a with defining polynomial x^2 - 109,
 Number Field in a with defining polynomial x^3 - x^2 - 36*x + 4,
 Number Field in a with defining polynomial x^4 - x^3 + 13*x^2 - 19*x + 361,
 Number Field in a with defining polynomial x^4 - x^3 + 14*x^2 + 34*x + 393,
 Number Field in a with defining polynomial x^5 + x^4 - 6*x^3 - x^2 + 18*x +␣
↪4,
 Number Field in a with defining polynomial x^5 + 2*x^4 + 7*x^3 + 4*x^2 +␣
↪11*x - 6,
 Number Field in a with defining polynomial x^5 - x^4 - 40*x^3 - 93*x^2 -␣
↪21*x + 17]
```

```
>>> from sage.all import *
>>> J = JonesDatabase()              # optional - database_jones_numfield
>>> J.unramified_outside([Integer(101),Integer(109)]) # optional - database_
↪jones_numfield
[Number Field in a with defining polynomial x - 1,
 Number Field in a with defining polynomial x^2 - 101,
 Number Field in a with defining polynomial x^2 - 109,
 Number Field in a with defining polynomial x^3 - x^2 - 36*x + 4,
 Number Field in a with defining polynomial x^4 - x^3 + 13*x^2 - 19*x + 361,
 Number Field in a with defining polynomial x^4 - x^3 + 14*x^2 + 34*x + 393,
 Number Field in a with defining polynomial x^5 + x^4 - 6*x^3 - x^2 + 18*x +␣
↪4,
 Number Field in a with defining polynomial x^5 + 2*x^4 + 7*x^3 + 4*x^2 +␣
↪11*x - 6,
 Number Field in a with defining polynomial x^5 - x^4 - 40*x^3 - 93*x^2 -␣
↪21*x + 17]
```

sage.databases.jones.**sortkey**(*K*)

A completely deterministic sorting key for number fields.

EXAMPLES:

```
sage: from sage.databases.jones import sortkey
sage: sortkey(QuadraticField(-3))
(2, 3, False, x^2 + 3)
```

```
>>> from sage.all import *
>>> from sage.databases.jones import sortkey
>>> sortkey(QuadraticField(-Integer(3)))
(2, 3, False, x^2 + 3)
```

2.8 Database of Hilbert polynomials

This module gives access to the database of Hilbert class polynomials. To use the database, you need to install the optional database_kohel package by the Sage command

```
sage -i database_kohel
```

EXAMPLES:

```
sage: # optional - database_kohel
sage: db = HilbertClassPolynomialDatabase()
sage: db[32]
x^2 - 52250000*x + 12167000000
```

```
>>> from sage.all import *
>>> # optional - database_kohel
>>> db = HilbertClassPolynomialDatabase()
>>> db[Integer(32)]
x^2 - 52250000*x + 12167000000
```

AUTHORS:

- David Kohel (2006-08-04): initial version

class sage.databases.db_class_polynomials.**AtkinClassPolynomialDatabase**

Bases: *ClassPolynomialDatabase*

The database of Atkin class polynomials.

model = 'Atk'

class sage.databases.db_class_polynomials.**ClassPolynomialDatabase**

Bases: object

class sage.databases.db_class_polynomials.**DedekindEtaClassPolynomialDatabase**

Bases: *ClassPolynomialDatabase*

The database of Dedekind eta class polynomials.

model = 'Eta'

class sage.databases.db_class_polynomials.**HilbertClassPolynomialDatabase**

Bases: *ClassPolynomialDatabase*

The database of Hilbert class polynomials.

EXAMPLES:

```
sage: # optional - database_kohel
sage: db = HilbertClassPolynomialDatabase()
sage: db[-4]
```

(continues on next page)

(continued from previous page)

```
x - 1728
sage: db[-7]
x + 3375
sage: f = db[-23]; f
x^3 + 3491750*x^2 - 5151296875*x + 12771880859375
sage: f.discriminant().factor()
-1 * 5^18 * 7^12 * 11^4 * 17^2 * 19^2 * 23
sage: db[-23]
x^3 + 3491750*x^2 - 5151296875*x + 12771880859375
```

```
>>> from sage.all import *
>>> # optional - database_kohel
>>> db = HilbertClassPolynomialDatabase()
>>> db[-Integer(4)]
x - 1728
>>> db[-Integer(7)]
x + 3375
>>> f = db[-Integer(23)]; f
x^3 + 3491750*x^2 - 5151296875*x + 12771880859375
>>> f.discriminant().factor()
-1 * 5^18 * 7^12 * 11^4 * 17^2 * 19^2 * 23
>>> db[-Integer(23)]
x^3 + 3491750*x^2 - 5151296875*x + 12771880859375
```

model = 'Cls'

class sage.databases.db_class_polynomials.**WeberClassPolynomialDatabase**

Bases: *ClassPolynomialDatabase*

The database of Weber class polynomials.

2.9 Database of modular polynomials

This module gives access to the database of modular polynomials. To use the database, you need to install the optional database_kohel package by the Sage command

```
sage -i database_kohel
```

EXAMPLES:

```
sage: # optional - database_kohel
sage: DBMP = ClassicalModularPolynomialDatabase()
sage: f = DBMP[29]
sage: f.degree()
58
sage: f.coefficient([28,28])
400152899204646997840260839128
```

```
>>> from sage.all import *
>>> # optional - database_kohel
>>> DBMP = ClassicalModularPolynomialDatabase()
>>> f = DBMP[Integer(29)]
```

(continues on next page)

(continued from previous page)

```
>>> f.degree()
58
>>> f.coefficient([Integer(28),Integer(28)])
400152899204646997840260839128
```

AUTHORS:

- David Kohel (2006-08-04): initial version

class sage.databases.db_modular_polynomials.**AtkinModularCorrespondenceDatabase**

Bases: *ModularCorrespondenceDatabase*

model = 'AtkCrr'

class sage.databases.db_modular_polynomials.**AtkinModularPolynomialDatabase**

Bases: *ModularPolynomialDatabase*

The database of modular polynomials Phi(x,j) for $X_0(p)$, where x is a function on invariant under the Atkin-Lehner invariant, with pole of minimal order at infinity.

model = 'Atk'

class sage.databases.db_modular_polynomials.**ClassicalModularPolynomialDatabase**

Bases: *ModularPolynomialDatabase*

The database of classical modular polynomials, i.e. the polynomials Phi_N(X,Y) relating the j-functions j(q) and j(q^N).

model = 'Cls'

class sage.databases.db_modular_polynomials.**DedekindEtaModularCorrespondenceDatabase**

Bases: *ModularCorrespondenceDatabase*

The database of modular correspondences in $X_0(p) \times X_0(p)$, where the model of the curves $X_0(p) = \mathbf{P}^1$ are specified by quotients of Dedekind's eta function.

model = 'EtaCrr'

class sage.databases.db_modular_polynomials.**DedekindEtaModularPolynomialDatabase**

Bases: *ModularPolynomialDatabase*

The database of modular polynomials Phi_N(X,Y) relating a quotient of Dedekind eta functions, well-defined on X_0(N), relating x(q) and the j-function j(q).

model = 'Eta'

class sage.databases.db_modular_polynomials.**ModularCorrespondenceDatabase**

Bases: *ModularPolynomialDatabase*

class sage.databases.db_modular_polynomials.**ModularPolynomialDatabase**

Bases: `object`

2.10 Database of the zeros of the Riemann zeta function

The main access function to the database of the zeros of the Riemann zeta function is *zeta_zeros()*. In order to use `zeta_zeros()`, you need to install the optional database_odlyzko_zeta package:

```
sage -i database_odlyzko_zeta
```

AUTHORS:

- William Stein: initial version
- Jeroen Demeyer (2015-01-20): converted `database_odlyzko_zeta` to new-style package

`sage.databases.odlyzko.`**`zeta_zeros`**`()`

List of the imaginary parts of the first 2,001,052 zeros of the Riemann zeta function, accurate to within 4e-9.

REFERENCES:

- http://www.dtc.umn.edu/~odlyzko/zeta_tables/index.html

EXAMPLES:

The following example shows the imaginary part of the 13th nontrivial zero of the Riemann zeta function:

```
sage: # optional - database_odlyzko_zeta
sage: zz = zeta_zeros()
sage: zz[12]
59.347044003
sage: len(zz)
2001052
```

```
>>> from sage.all import *
>>> # optional - database_odlyzko_zeta
>>> zz = zeta_zeros()
>>> zz[Integer(12)]
59.347044003
>>> len(zz)
2001052
```

CHAPTER

THREE

INDICES AND TABLES

- Index
- Module Index
- Search Page

PYTHON MODULE INDEX

d

INDEX

Non-alphabetical

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

Z