Sage Reference Manual: Databases

Release 7.1

The Sage Development Team

March 20, 2016

CONTENTS

1 Cremona's tables of elliptic curves **3**

2 The Stein-Watkins table of elliptic curves **17**

3 John Jones's tables of number fields **21**

4 The On-Line Encyclopedia of Integer Sequences (OEIS) **25**
4.1 Classes and methods 27

5 Local copy of Sloane On-Line Encyclopedia of Integer Sequences **45**
5.1 Classes and methods 46

6 FindStat - the Combinatorial Statistic Finder. **49**
6.1 A guided tour 49
6.2 Classes and methods 51

7 Frank Luebeck's tables of Conway polynomials over finite fields **69**

8 Tables of zeros of the Riemann-Zeta function **71**

9 Ideals from the Symbolic Data project **73**

10 Cunningham table **75**

11 Database of Hilbert Polynomials **77**

12 Database of Modular Polynomials **79**

13 Indices and Tables **81**

Bibliography **83**

There are numerous specific mathematical databases either included in Sage or available as optional packages. Also, Sage includes two powerful general database packages.

Sage includes the ZOPE object oriented database ZODB, which "is a Python object persistence system. It provides transparent object-oriented persistency."

Sage also includes the powerful relational database SQLite, along with a Python interface to SQLite. SQlite is a small C library that implements a self-contained, embeddable, zero-configuration SQL database engine.

- Transactions are atomic, consistent, isolated, and durable (ACID) even after system crashes and power failures.
- Zero-configuration - no setup or administration needed.
- Implements most of SQL92. (Features not supported)
- A complete database is stored in a single disk file.
- Database files can be freely shared between machines with different byte orders.
- Supports databases up to 2 tebibytes (2^41 bytes) in size.
- Strings and BLOBs up to 2 gibibytes (2^31 bytes) in size.
- Small code footprint: less than 250KiB fully configured or less than 150KiB with optional features omitted.
- Faster than popular client/server database engines for most common operations.
- Simple, easy to use API.
- TCL bindings included. Bindings for many other languages available separately.
- Well-commented source code with over 95% test coverage.
- Self-contained: no external dependencies.
- Sources are in the public domain. Use for any purpose.

CHAPTER

ONE

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, run the following 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
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*, *set_global=None*)

Initializes 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`.

If you would like other components of Sage to use this database, mark `set_global=True`.

TESTS:

```
sage: c = CremonaDatabase()
sage: isinstance(c, sage.databases.cremona.MiniCremonaDatabase)
True
sage: isinstance(c, sage.databases.cremona.LargeCremonaDatabase)  # optional - database_cremona_
True
```

Verify that trac ticket #12341 has been resolved:

```
sage: c = CremonaDatabase('should not exist',mini=True)
Traceback (most recent call last):
...
ValueError: Desired database (='should not exist') does not exist
sage: c = CremonaDatabase('should not exist',mini=False)
Traceback (most recent call last):
...
ValueError: Desired database (='should not exist') does not exist
sage: from sage.env import SAGE_SHARE
sage: os.path.isfile(os.path.join(SAGE_SHARE,'cremona','should_not_exist.db'))
False
```

class `sage.databases.cremona.`**`LargeCremonaDatabase`**(*name*, *read_only=True*, *build=False*)

Bases: `sage.databases.cremona.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]}
```

`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` - int, the conductor

OUTPUT: dict 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]
```

`allgens`(*N*)

Return the allgens table for conductor N.

INPUT:

- `N` - int, the conductor

OUTPUT:

- `dict` - 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]]
```

degphi(*N*)

Return the degphi table for conductor N.

INPUT:

- `N` - int, the conductor

OUTPUT:

- `dict` - 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
```

class sage.databases.cremona.**MiniCremonaDatabase**(*name*, *read_only=True*, *build=False*)

Bases: `sage.databases.sql_db.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]}
```

allcurves(*N*)

Returns the allcurves table of curves of conductor N.

INPUT:

- `N` - int, the conductor

OUTPUT:

- `dict` - 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]
```

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
```

Check that trac ticket #17904 is fixed:

```
sage: 'gens' in CremonaDatabase().coefficients_and_data('100467a2')[1] # optional - database
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)
```

curves(*N*)

Returns the curves table of all *optimal* curves of conductor N.

INPUT:

- `N` - int, the conductor

OUTPUT:

- `dict` - 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]}
```

Note the 'h3', which is the unique case in the tables where the optimal curve doesn't have label ending in 1:

```
sage: list(sorted(CremonaDatabase().curves(990).keys()))
['a1', 'b1', 'c1', 'd1', 'e1', 'f1', 'g1', 'h3', 'i1', 'j1', 'k1', 'l1']
```

TESTS:

```
sage: c = CremonaDatabase()
sage: c.curves(12001)['a1']    # optional - database_cremona_ellcurve
[[1, 0, 0, -101, 382], 1, 1]
```

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
```

Check that trac ticket #17904 is fixed:

```
sage: ai = EllipticCurve('100467a2').ainvs() # optional - database_cremona_ellcurve
sage: 'gens' in CremonaDatabase().data_from_coefficients(ai) # optional - database_cremona_e
True
```

elliptic_curve(*label*)

Return an elliptic curve with given label with some data about it from the database pre-filled in.

INPUT:

- `label` - str (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
```

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
```

elliptic_curve_from_ainvs(*ainvs*)

Returns the elliptic curve in the database of with minimal ainvs, if it exists, or 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_ellcu
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
```

Make sure trac ticket #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
```

isogeny_class (*label*)

Returns the isogeny class of elliptic curves that are isogenous to the curve with given Cremona label.

INPUT:

- `label` - string

OUTPUT:

- `list` - 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]
```

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]]]
```

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']
```

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']
```

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', '
```

largest_conductor ()

The largest conductor for which the database is complete.

OUTPUT:

- `int` - largest conductor

EXAMPLES:

```
sage: c = CremonaDatabase('cremona mini')
sage: c.largest_conductor()
9999
```

list (*conductors*)

Returns 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]
```

list_optimal (*conductors*)

Returns 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]
```

number_of_curves (*N=0*, *i=0*)

Returns 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 - int
- i - int or str

OUTPUT: int

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()
```

number_of_isogeny_classes (*N=0*)

Returns 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 - int

OUTPUT: int

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()
```

random ()

Returns 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
```

smallest_conductor()

The smallest conductor for which the database is complete: always 1.

OUTPUT:

- `int` - 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
```

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
```

sage.databases.cremona.**class_to_int**(*k*)

Converts 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'
```

sage.databases.cremona.**cmp_code**(*key1*, *key2*)

Comparison function for curve id strings.

Note: Not the same as standard lexicographic order!

EXAMPLES:

```
sage: import sage.databases.cremona as cremona
sage: cremona.cmp_code('ba1','z1')
1
```

By contrast:

```
sage: cmp('ba1','z1')
-1
```

sage.databases.cremona.**cremona_letter_code**(*n*)

Returns 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` (int) – a non-negative integer

OUTPUT: str

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'
```

TESTS:

```
sage: cremona_letter_code(QQ)
Traceback (most recent call last):
...
ValueError: Cremona letter codes are only defined for non-negative integers
sage: cremona_letter_code(x)
Traceback (most recent call last):
...
ValueError: Cremona letter codes are only defined for non-negative integers
sage: cremona_letter_code(-1)
Traceback (most recent call last):
...
ValueError: Cremona letter codes are only defined for non-negative integers
sage: cremona_letter_code(3.14159)
Traceback (most recent call last):
...
ValueError: Cremona letter codes are only defined for non-negative integers
```

`sage.databases.cremona.`**`cremona_to_lmfdb`**(*cremona_label*, *CDB=None*)

Converts a Cremona label into an LMFDB label.

See `parse_lmfdb_label()` for an explanation of LMFDB labels.

INPUT:

- `cremona_label` – a 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` – a 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'
```

TESTS:

```
sage: for label in ['5077a1','66a3','102b','420c2']:
...       assert(lmfdb_to_cremona(cremona_to_lmfdb(label)) == label)
sage: for label in ['438.c2','306.b','462.f3']:
...       assert(cremona_to_lmfdb(lmfdb_to_cremona(label)) == label)
```

`sage.databases.cremona.`**`is_optimal_id`**(*id*)

Returns 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` - str of form letter code followed by an integer, e.g., a3, bb5, etc.

OUTPUT: bool

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
```

`sage.databases.cremona.`**`lmfdb_to_cremona`**(*lmfdb_label*, *CDB=None*)

Converts an LMFDB labe into a Cremona label.

See `parse_lmfdb_label()` for an explanation of LMFDB labels.

INPUT:

- `lmfdb_label` – a 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'
```

`sage.databases.cremona.`**`old_cremona_letter_code`**(*n*)

Returns the *old* Cremona letter code corresponding to an integer. integer.

For example:

```
1  --> A
26 --> Z
27 --> AA
52 --> ZZ
53 --> AAA
etc.
```

INPUT:

- `n` - int

OUTPUT: str

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'
```

`sage.databases.cremona.`**`parse_cremona_label`**(*label*)

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` - str

OUTPUT:

- `int` - the conductor
- `str` - the isogeny class label
- `int` - 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)
```

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
```

TESTS:

```
sage: from sage.databases.cremona import parse_cremona_label
sage: parse_cremona_label('x11')
Traceback (most recent call last):
...
ValueError: x11 is not a valid Cremona label
```

sage.databases.cremona.**parse_lmfdb_label**(*label*)

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

OUTPUT:

- `int` - the conductor
- `str` - 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)
```

`sage.databases.cremona.`**`split_code`**`(`*key*`)`

Splits class+curve id string into its two parts.

EXAMPLES:

```
sage: import sage.databases.cremona as cremona
sage: cremona.split_code('ba2')
('ba', '2')
```

CHAPTER TWO

THE STEIN-WATKINS TABLE OF ELLIPTIC CURVES

Sage gives access to the Stein-Watkins table of elliptic curves, via an optional 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, 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 [SteinWatkins]).

- 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 $n10^6$ and $(n+1)10^6$. 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
```

We type `next(d)` to get each isogeny class of curves from `d`:

```
sage: C = next(d)                                    # optional - database_stein_watkins
sage: C                                              # optional - database_stein_watkins
Stein-Watkins isogeny class of conductor 11
sage: next(d)                                        # optional - database_stein_watkins
Stein-Watkins isogeny class of conductor 14
sage: next(d)                                        # optional - database_stein_watkins
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: C.data                                          # optional - database_stein_watkins
['11', '[11]', '0', '0.253842', '25', '+*1']
sage: C.curves                                        # optional - database_stein_watkins
[[[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                                        # optional - database_stein_watkins
11
sage: C.leading_coefficient                              # optional - database_stein_watkins
'0.253842'
sage: C.modular_degree                                   # optional - database_stein_watkins
'+*1'
sage: C.rank                                             # optional - database_stein_watkins
0
sage: C.isogeny_number                                   # optional - database_stein_watkins
'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']]
```

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
```

Each call `next(d)` gives another elliptic curve of prime conductor:

```
sage: C = next(d)                                      # optional - database_stein_watkins
sage: C                                                # optional - database_stein_watkins
Stein-Watkins isogeny class of conductor 17
sage: C.curves                                         # optional - database_stein_watkins
[[[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)                                      # optional - database_stein_watkins
sage: C                                                # optional - database_stein_watkins
Stein-Watkins isogeny class of conductor 19
```

REFERENCE:

class `sage.databases.stein_watkins.`**`SteinWatkinsAllData`**`(`*num*`)`

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.

EXAMPLE:

```
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]
```

next()

class sage.databases.stein_watkins.**SteinWatkinsIsogenyClass**(*conductor*)

class sage.databases.stein_watkins.**SteinWatkinsPrimeData**(*num*)

Bases: sage.databases.stein_watkins.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, 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]
```

CHAPTER

THREE

JOHN JONES'S TABLES OF NUMBER FIELDS

In order to use the Jones database, the optional database 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
```

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_num
[(1, 1), (2, -4), (2, -8), (2, 8), (4, 256), (4, 512), (4, -1024), (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_num
[-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_num
[-135, -675, -6075, -6075]
sage: factor(6075)
3^5 * 5^2
sage: factor(675)
3^3 * 5^2
sage: factor(135)
3^3 * 5
```

List all fields in the database ramified at 101:

```
sage: J.ramified_at(101)                                              # optional - database_jones_num
[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 + 2*x^4 + 7*x^3 + 4*x^2 + 11*x - 6,
 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 - x^4 - 40*x^3 - 93*x^2 - 21*x + 17]
```

class `sage.databases.jones.`**`JonesDatabase`**

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
```

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: J = JonesDatabase()              # optional - database_jones_numfield
sage: J.ramified_at([101,109])         # optional - database_jones_numfield
[]
sage: J.ramified_at([109])             # optional - database_jones_numfield
[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)               # 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 + 2*x^4 + 7*x^3 + 4*x^2 + 11*x - 6,
 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 - x^4 - 40*x^3 - 93*x^2 - 21*x + 17]
sage: J.ramified_at((2, 5, 29), 3, 'c') # optional - database_jones_numfield
[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 + 97*x - 333,
 Number Field in c with defining polynomial x^3 - x^2 - 48*x - 188]
```

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 + 2*x^4 + 7*x^3 + 4*x^2 + 11*x - 6,
 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 - x^4 - 40*x^3 - 93*x^2 - 21*x + 17]
```

CHAPTER

FOUR

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.

AUTHORS:

- Thierry Monteil (2012-02-10 – 2013-06-21): initial version.
- Vincent Delecroix (2014): modifies continued fractions because of trac ticket #14567

EXAMPLES:

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

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

```
sage: search = oeis([3, 7, 15, 1], max_results=4) ; search  # optional -- internet
0: A001203: Continued fraction expansion of Pi.
1: A082495: a(n) = (2^n - 1) mod n.
2: A165416: Irregular array read by rows: The n-th row contains those distinct positive integers that
3: A246674: Run Length Transform of A000225.

sage: [u.id() for u in search]                           # optional -- internet
['A001203', 'A082495', 'A165416', 'A246674']
sage: c = search[0] ; c                                  # optional -- internet
A001203: Continued fraction expansion of Pi.


sage: c.first_terms(15)                                  # optional -- internet
(3, 7, 15, 1, 292, 1, 1, 1, 2, 1, 3, 1, 14, 2, 1)

sage: 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, ...]

sage: c.comments()                                       # optional -- internet
0: The first 5,821,569,425 terms were computed by _Eric W. Weisstein_ on Sep 18 2011.
1: The first 10,672,905,501 terms were computed by _Eric W. Weisstein_ on Jul 17 2013.
2: The first 15,000,000,000 terms were computed by _Eric W. Weisstein_ on Jul 27 2013.


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

```
sage: x.convergents()[:7]                       # optional -- internet
[3, 22/7, 333/106, 355/113, 103993/33102, 104348/33215, 208341/66317]

sage: RR(x.value())                             # optional -- internet
3.14159265358979
sage: RR(x.value()) == RR(pi)                   # optional -- internet
True
```

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

```
sage: [Posets(i).cardinality() for i in range(10)]
[1, 1, 2, 5, 16, 63, 318, 2045, 16999, 183231]
sage: oeis(_)                                   # optional -- internet
0: A000112: Number of partially ordered sets ("posets") with n unlabeled elements.
sage: p = _[0]                                  # optional -- internet

sage: 'hard' in p.keywords()                    # optional -- internet
True
sage: len(p.formulas())                         # optional -- internet
0
sage: len(p.first_terms())                      # optional -- internet
17

sage: p.cross_references(fetch=True)            # optional -- internet
0: A000798: Number of different quasi-orders (or topologies, or transitive digraphs) with n labeled e
1: A001035: Number of partially ordered sets ("posets") with n labeled elements (or labeled acyclic t
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.
```

What does the Taylor expansion of the $e^(e^x - 1)`$ function have to do with primes ?

```
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)                                   # optional -- internet
0: A000110: Bell or exponential numbers: number of ways to partition a set of n labeled elements.

sage: b = _[0]                                  # optional -- internet

sage: b.formulas()[0]                           # optional -- internet
'E.g.f.: exp(exp(x) - 1).'

sage: [i for i in b.comments() if 'prime' in i][-1]      # optional -- internet
'Number n is prime if mod(a(n)-2,n) = 0. -_Dmitry Kruchinin_, Feb 14 2012'

sage: [n for n in range(2, 20) if (b(n)-2) % n == 0]     # optional -- internet
[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.

Todo

- in case of flood, suggest the user to install the off-line database instead.
- interface with the off-line database (or reimplement it).

4.1 Classes and methods

class `sage.databases.oeis.`**`FancyTuple`**

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'
```

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

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 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 (http://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.
```

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: A027926: Triangular array T read by rows: T(n,0) = T(n,2n) = 1 for n >= 0; T(n,1) = 1 for n >
2: A001129: Iccanobif numbers: reverse digits of two previous terms and add.

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: 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)

sage: fibo.cross_references()[0]                # optional -- internet
'A039834'

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: sfibo.first_terms(absolute_value=True)[2:20] == fibo.first_terms()[:18]   # optional -- in
True

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

sage: fibo.comments()[1]                        # optional -- internet
"F(n+2) = number of binary sequences of length n that have no
consecutive 0's."

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

The database can be searched by description:

```
sage: oeis('prime gap factorization', max_results=4)                # optional -- internet
0: A073491: Numbers having no prime gaps in their factorization.
```

```
1: A073490: Number of prime gaps in factorization of n.
2: A073492: Numbers having at least one prime gap in their factorization.
3: A073493: Numbers having exactly one prime gap in their factorization.
```

Warning: The following will fetch the OEIS database twice (once for searching the database, and once again for creating the sequence `fibo`):

```
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: A027926: Triangular array T read by rows: T(n,0) = T(n,2n) = 1 for n >= 0; T(n,1) = 1 for n
2: A001129: Iccanobif numbers: reverse digits of two previous terms and add.

sage: fibo = oeis('A000045')                # optional -- internet
```

Do not do this, it is slow, it costs bandwidth and server resources ! Instead, do the following, to reuse the result of the search to create the sequence:

```
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: A027926: Triangular array T read by rows: T(n,0) = T(n,2n) = 1 for n >= 0; T(n,1) = 1 for n
2: A001129: Iccanobif numbers: reverse digits of two previous terms and add.

sage: fibo = _[0]                           # optional -- internet
```

browse()

Open the OEIS web page in a browser.

EXAMPLES:

```
sage: 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: A073491: Numbers having no prime gaps in their factorization.
1: A073490: Number of prime gaps in factorization of n.
2: A073492: Numbers having at least one prime gap in their factorization.

sage: prime_gaps = _[1] ; prime_gaps        # optional -- internet
A073490: Number of prime gaps in factorization of n.
```

```
sage: oeis('beaver')                            # optional -- internet
0: A028444: Busy Beaver sequence, or Rado's sigma function: ...
1: A060843: Busy Beaver problem: a(n) = maximal number of steps ...
2: A131956: Busy Beaver variation: maximum number of steps for ...

sage: oeis('beaver', max_results=4, first_result=2)      # optional -- internet
0: A131956: Busy Beaver variation: maximum number of steps for ...
1: A141475: Number of Turing machines with n states following ...
2: A131957: Busy Beaver sigma variation: maximum number of 1's ...
3: A052200: Number of n-state, 2-symbol, d+ in {LEFT, RIGHT}, ...
```

find_by_id(*ident*)

INPUT:

- `ident` - a string representing the A-number of the sequence or an integer representing its number.

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.
```

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

Search for OEIS sequences containing the given subsequence.

INPUT:

- `subsequence` - a list 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 -
0: A000045: Fibonacci numbers: F(n) = F(n-1) + F(n-2) with F(0) = 0 and F(1) = 1.
1: A177194: Fibonacci numbers whose decimal expression does not contain any digit 0.
2: A212804: Expansion of (1-x)/(1-x-x^2).

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.
```

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

Bases: `sage.structure.sage_object.SageObject`

The class of OEIS sequences.

This class implements OEIS sequences. Such sequences are produced from a string in the OEIS format. 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: sfibo = oeis('A039834')                # optional -- internet
sage: sfibo.first_terms()[:10]               # optional -- internet
(1, 1, 0, 1, -1, 2, -3, 5, -8, 13)

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

sage: sfibo[0]                               # optional -- internet
1
sage: sfibo[6]                               # optional -- internet
-3
```

`__call__`(*k*)

Returns the element of the sequence `self` whith 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

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

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

TESTS:

```
sage: s = oeis._imaginary_sequence()
sage: s(38)
1
sage: s(42)
-1
sage: s(2)
Traceback (most recent call last):
...
ValueError: Sequence A999999 is not defined (or known) for index 2
```

author()

Returns 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_, Apr 30 1991'
```

TESTS:

```
sage: s = oeis._imaginary_sequence()
sage: s.author()
'Anonymous.'
```

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
```

TESTS:

```
sage: s = oeis._imaginary_sequence()            # optional -- webbrowser
sage: s.browse()                                # optional -- 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()[:3]                          # optional -- internet
0: Also sometimes called Lamé's sequence.
1: F(n+2) = number of binary sequences of length n that have no consecutive 0's.
2: F(n+2) = number of subsets of {1,2,...,n} that contain no consecutive integers.
```

TESTS:

```
sage: s = oeis._imaginary_sequence()
sage: s.comments()
0: 42 is the product of the first 4 prime numbers, except 5 and perhaps 1.
1: Apart from that, i have no comment.
```

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` if `True`, return a tuple of OEIS sequences.

EXAMPLES:

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

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

sage: nbalanced.cross_references(fetch=True)    # optional -- internet
0: A049703: a(0) = 0; for n>0, a(n) = A005598(n)/2.
1: A049695: Array T read by diagonals; T(i,j)=number of nonnegative slopes of lines determin
2: A103116: A005598(n) - 1.
3: A000010: Euler totient function phi(n): count numbers <= n and prime to n.

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

TESTS:

```
sage: s = oeis._imaginary_sequence()
sage: s.cross_references()
('A000042', 'A000024')
```

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: Continued fraction expansion of Pi.

sage: 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, ...]
```

TESTS:

```
sage: s = oeis._imaginary_sequence()
sage: s.examples()
0: s(42) + s(43) = 0.
```

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); or Fibonacci numbers (A000045) ex

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

TESTS:

```
sage: s = oeis._imaginary_sequence()
sage: s.extensions_or_errors()
0: This sequence does not contain errors.
```

first_terms(*number=None*, *absolute_value=False*)

INPUT:

- `number` - (integer or `None`, default: `None`) the number of terms returned (if less than the number of available terms). When set to None, returns all the known terms.
- `absolute_value` - (bool, default: `False`) when a sequence has negative entries, OEIS also stores the absolute values of its first terms, when `absolute_value` is set to `True`, you will get them.

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)
```

TESTS:

```
sage: s = oeis._imaginary_sequence()
sage: s.first_terms()
(1, 1, 1, 1, -1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1)
sage: s.first_terms(5)
(1, 1, 1, 1, -1)
sage: s.first_terms(5, absolute_value=True)
(1, 1, 1, 1, 1)

sage: s = oeis._imaginary_sequence(keywords='full')
sage: s(40)
Traceback (most recent call last):
...
TypeError: You found a sign inconsistency, please contact OEIS
```

```
sage: s = oeis._imaginary_sequence(keywords='sign,full')
sage: s(40)
1

sage: s = oeis._imaginary_sequence(keywords='nonn,full')
sage: s(42)
1
```

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)).'
```

TESTS:

```
sage: s = oeis._imaginary_sequence()
sage: s.formulas()
0: For n big enough, s(n+1) - s(n) = 0.
```

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'

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

TESTS:

```
sage: s = oeis._imaginary_sequence()
sage: s.id()
'A999999'
sage: s.id(format='int')
999999
```

is_finite()

Tells 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:

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

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
```

TESTS:

```
sage: s = oeis._imaginary_sequence()
sage: s.is_finite()
Unknown

sage: s = oeis._imaginary_sequence('nonn,finit')
sage: s.is_finite()
True
```

is_full()

Tells 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:

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

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
```

TESTS:

```
sage: s = oeis._imaginary_sequence()
sage: s.is_full()
Unknown

sage: s = oeis._imaginary_sequence('nonn,full,finit')
sage: s.is_full()
True
```

keywords()

Return the keywords associated to the sequence `self`.

OUTPUT:

- tuple of strings.

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.keywords()                              # optional -- internet
('core', 'nonn', 'nice', 'easy', 'hear')
```

TESTS:

```
sage: s = oeis._imaginary_sequence()
sage: s.keywords()
('sign', 'easy')

sage: s = oeis._imaginary_sequence(keywords='nonn,hard')
sage: s.keywords()
('nonn', 'hard')
```

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

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

INPUT:

- `browse` - an 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: http://oeis.org/A000045/b000045.txt
```

```
1: http://www.schoolnet.ca/vp-pv/amof/e_fiboI.htm
...

sage: f.links(format='raw')                    # optional -- internet
0: N. J. A. Sloane, <a href="/A000045/b000045.txt">The first 2000 Fibonacci numbers: Table o
1: Amazing Mathematical Object Factory, <a href="http://www.schoolnet.ca/vp-pv/amof/e_fiboI.
...
```

TESTS:

```
sage: s = oeis._imaginary_sequence()
sage: s.links(format='raw')[2]
'Do not confuse with the sequence <a href="/A000042">A000042</a> or the sequence <a href="/A

sage: s.links(format='url')[3]
'http://oeis.org/A000024'

sage: HTML = s.links(format="html");  HTML
0: Wikipedia, <a href="http://en.wikipedia.org/wiki/42_(number)">42 (number)</a>
1: See. also <a href="http://trac.sagemath.org/sage_trac/ticket/42">trac ticket #42</a>
...
sage: type(HTML)
<class 'sage.misc.html.HtmlFragment'>
```

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.'
```

TESTS:

```
sage: s = oeis._imaginary_sequence()
sage: s.name()
'The opposite of twice the characteristic sequence of 42 plus one, starting from 38.'
```

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 http://oeis.org/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)
sage: x[:90] == cfg[:90]                     # optional -- internet
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
True

sage: av = oeis('A087778') ; av              # optional -- internet
A087778: Decimal expansion of Avogadro's constant.

sage: av.natural_object()                    # optional -- internet
6.022141000000000?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

sage: sfib = oeis('A039834') ; sfib          # optional -- internet
A039834: a(n+2) = -a(n+1)+a(n) (signed Fibonacci numbers); or Fibonacci numbers (A000045) ex

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

TESTS:

```
sage: s = oeis._imaginary_sequence('nonn,cofr')
sage: type(s.natural_object())
<class 'sage.rings.continued_fraction.ContinuedFraction_periodic'>

sage: s = oeis._imaginary_sequence('nonn')
sage: s.natural_object().universe()
Non negative integer semiring
```

```
sage: s = oeis._imaginary_sequence()
sage: s.natural_object().universe()
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)
```

TESTS:

```
sage: s = oeis._imaginary_sequence()
sage: s.offsets()
(38, 4)
```

old_IDs()

Returns 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')
```

TESTS:

```
sage: s = oeis._imaginary_sequence()
sage: s.old_IDs()
('M9999', 'N9999')
```

programs(*language='other'*)

Returns programs implementing the sequence `self` in the given `language`.

INPUT:

- `language` - string (default: 'other') - the language of the program. Current values are: 'maple', 'mathematica' and 'other'.

OUTPUT:

- tuple of strings (with fancy formatting).

Todo

ask OEIS to add a "Sage program" field in the database ;)

EXAMPLES:

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

sage: ee.programs()[0]                          # optional -- internet
'(PARI) { default(realprecision, 50080); x=exp(1); for (n=1, 50000, d=floor(x); x=(x-d)*10;
```

TESTS:

```
sage: s = oeis._imaginary_sequence()
sage: s.programs()
0: (Python)
1: def A999999(n):
2:     assert(isinstance(n, (int, Integer))), "n must be an integer."
3:     if n < 38:
4:         raise ValueError("The value %s is not accepted." %str(n)))
5:     elif n == 42:
6:         return -1
7:     else:
8:         return 1

sage: s.programs('maple')
0: Do not even try, Maple is not able to produce such a sequence.

sage: s.programs('mathematica')
0: Mathematica neither.
```

raw_entry()

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

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,...
...
```

TESTS:

```
sage: s = oeis._imaginary_sequence()
sage: s.raw_entry() == oeis._imaginary_entry('sign,easy')
True
```

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
0: A. H. Beiler, Recreations in the Theory of Numbers, Dover, NY, 1964, p. 52.
1: C. Clawson, Mathematical Mysteries, Plenum Press, 1996, p. 180.
2: R. Crandall and C. Pomerance, Prime Numbers: A Computational Perspective, Springer, NY, 2
3: G. H. Hardy and E. M. Wright, An Introduction to the Theory of Numbers, 5th ed., Oxford U
...

sage: _[0]                                  # optional -- internet
'A. H. Beiler, Recreations in the Theory of Numbers, Dover, NY, 1964, p. 52.'
```

TESTS:

```
sage: s = oeis._imaginary_sequence()
sage: s.references()[1]
'Lewis Carroll, The Hunting of the Snark.'
```

show()

Display most available informations 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

FIRST TERMS
(2, 8, 248, 11328, 849312, 94857600, 14819214720, 3091936512000, 831657655349760, 2804737561

FORMULAS
...
OFFSETS
(0, 1)

URL
http://oeis.org/A012345

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

TESTS:

```
sage: s = oeis._imaginary_sequence()
sage: s.show()
ID
A999999
```

```
NAME
The opposite of twice the characteristic sequence of 42 plus ...
FIRST TERMS
(1, 1, 1, 1, -1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...

COMMENTS
0: 42 is the product of the first 4 prime numbers, except ...
1: Apart from that, i have no comment.
...
```

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
'http://oeis.org/A000045'
```

TESTS:

```
sage: s = oeis._imaginary_sequence()
sage: s.url()
'http://oeis.org/A999999'
```

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

CHAPTER

FIVE

LOCAL COPY OF SLOANE 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]
```

To get the name of a sequence, type

```
sage: SloaneEncyclopedia.sequence_name(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,
```

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, ...
```

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's now also smarter about loading the local database in that it doesn't 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): deprecate dead code and update related doc and tests.

5.1 Classes and methods

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

A local copy of the Sloane Online Encyclopedia of Integer Sequences that contains only the sequence numbers and the sequence 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` - int

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

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

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

INPUT:

- `oeis_url` - string (default: "http://oeis.org...") The URL of the stripped.gz encyclopedia file.
- `names_url` - string (default: "http://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.

`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` - int

OUTPUT: string

EXAMPLES:

sage: SloaneEncyclopedia.sequence_name(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 verson. 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.

`sage.databases.sloane.`**`parse_sequence`**(*text=''*)

This internal function was only used by the sloane_find function, which is now deprecated.

TESTS:

```
sage: from sage.databases.sloane import parse_sequence
sage: parse_sequence()
doctest:...: DeprecationWarning: The function parse_sequence is not used anymore (2012-01-01).
See http://trac.sagemath.org/10358 for details.
```

`sage.databases.sloane.`**`sloane_find`**(*list=[ ]*, *nresults=30*, *verbose=True*)

This function is broken. It is replaced by the `OEIS` module.

Type `oeis?` for more information.

TESTS:

```
sage: sloane_find([1,2,3])
doctest:...: DeprecationWarning: The function sloane_find is deprecated. Use oeis() instead (201
See http://trac.sagemath.org/10358 for details.
```

`sage.databases.sloane.`**`sloane_sequence`**(*number=1*, *verbose=True*)

This function is broken. It is replaced by the `OEIS` module.

Type `oeis?` for more information.

TESTS:

```
sage: sloane_sequence(123)
doctest:...: DeprecationWarning: The function sloane_sequence is deprecated. Use oeis() instead
See http://trac.sagemath.org/10358 for details.
```

CHAPTER

SIX

FINDSTAT - THE COMBINATORIAL STATISTIC FINDER.

The FindStat database can be found at http://www.findstat.org .

Fix 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 sage interface to FindStat to:

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

To access the database, use `findstat`:

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

AUTHORS:

- Martin Rubey (2015): initial version.

6.1 A guided tour

6.1.1 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: PM8 = PerfectMatchings(8)
sage: r = findstat([(m, m.number_of_nestings()) for m in PM8]); r          # optional -- internet,ra
0: (St000041: The number of nestings of a perfect matching. , [], 105)
...
```

The result of this query is a list (presented as a `sage.databases.oeis.FancyTuple`) of triples. The first element of each triple is a `FindStatStatistic` $s : S \to \mathbf{Z}$, the second element a list of `FindStatMap`‘s $f_i : S_i \to S_{i+1}$, and the third element is an integer:

```
sage: (s, list_f, quality) = r[0]                                          # optional -- internet
```

The precise meaning of the result is as follows:

> The composition $f_n \circ ... \circ f_2 \circ f_1$ applied to the objects sent to FindStat agrees with $quality$ many $(object, value)$ pairs of s in the database. Moreover, there are no other $(object, value)$ pairs of s stored in the database, i.e., there is no disagreement of values.

Put differently, if $quality$ is not too small it is likely that the statistic sent to FindStat equals $s \circ f_n \circ ... \circ f_2 \circ f_1$.

In the case at hand, the list of maps is empty and the integer $quality$ equals the number of $(object, value)$ pairs passed to FindStat. This means, that the set of $(object, value)$ pairs of the statistic s as stored in the FindStat database is a superset of the data sent. We can now retrieve the description from the database:

```
sage: print s.description()                                          # optional -- internet,ra
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 $
```

and check the references:

```
sage: s.references()                                                 # optional -- internet,ra
0: [1] [[MathSciNet:1288802]]
1: [2] [[MathSciNet:1418763]]
```

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

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

Another interesting possibility is to look for equidistributed statistics. Instead of submitting a list of pairs, we pass a pair of lists:

```
sage: r = findstat((PM8, [m.number_of_nestings() for m in PM8])); r    # optional -- internet,ra
0: (St000041: The number of nestings of a perfect matching. , [], 105)
1: (St000042: The number of crossings of a perfect matching. , [], 105)
...
```

This results tells us that the database contains another entriy that is equidistributed with the number of nestings on perfect matchings of length 8, namely the number of crossings.

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(4), lambda pi: pi.saliances()[0]); r   # optional -- internet,ra
0: (St000051: The size of the left subtree. , [Mp00069: complement, Mp00061: to increasing tree], 24)
...
sage: (s, list_f, quality) = r[0]                                    # optional -- internet
```

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
```

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

```
sage: print s.description()                                          # optional -- internet,ra
The size of the left subtree.
```

```
sage: print s.code()                                              # optional -- internet,ra
def statistic(T):
    return T[0].node_number()

sage: print list_f[0].code() + "\r\n" + list_f[1].code()          # optional -- internet,ra
def complement(elt):
    n = len(elt)
    return elt.__class__(elt.parent(), map(lambda x: n - x + 1, elt) )

def increasing_tree_shape(elt, compare=min):
    return elt.increasing_tree(compare).shape()
```

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

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

6.1.2 Editing and submitting statistics

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

```
sage: findstat([(d, randint(1,1000)) for d in DyckWords(4)])      # optional -- internet
a new statistic on Cc0005: 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 `FindStatStatistic.set_description()`, `FindStatStatistic.set_code()` or `FindStatStatistic.set_references()`, and then `FindStatStatistic.submit()` your changes for review by the FindStat team.

6.2 Classes and methods

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

Bases: `sage.structure.sage_object.SageObject`

The Combinatorial Statistic Finder.

`FindStat` is a class representing results of queries to the FindStat database. This class is also the entry point to edit statistics and new submissions. Use the shorthand `findstat` to call it.

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.
- 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.
- 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 list of pairs of the form (list of strings, list of values), or a single pair of the form (list of strings, list of values). In each pair there should be as many strings as 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,r
St000041: The number of nestings of a perfect matching.

sage: findstat(51)                                           # optional -- internet,r
St000051: The size of the left subtree.
```

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

```
sage: q = findstat([(pi, pi.length()) for pi in Permutations(4)]); q    # optional -- internet,r
0: (St000018: The [[/Permutations/Inversions|number of inversions]] of a permutation., [], 24)
1: (St000004: The [[/Permutations/Descents-Major|major index]] of a permutation., [Mp00062: inve
...
```

or a dictionary:

```
sage: p = findstat({pi: pi.length() for pi in Permutations(4)}); p      # optional -- internet,r
0: (St000018: The [[/Permutations/Inversions|number of inversions]] of a permutation., [], 24)
1: (St000004: The [[/Permutations/Descents-Major|major index]] of a permutation., [Mp00062: inve
...
```

Note however, that the results of these two queries are not necessarily the same, because we compare queries by the data sent, and the ordering of the data might be different:

```
sage: p == q                                                 # optional -- internet
False
```

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("Permutations", lambda pi: pi.length())       # optional -- internet,r
0: (St000018: The [[/Permutations/Inversions|number of inversions]] of a permutation., [], 200)
...
```

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

```
sage: S4 = Permutations(4); findstat((S4, [pi.length() for pi in S4]))   # optional -- internet,r
0: (St000004: The [[/Permutations/Descents-Major|major index]] of a permutation., [], 24)
1: (St000018: The [[/Permutations/Inversions|number of inversions]] of a permutation., [], 24)
...
```

Note that there is a limit, `FINDSTAT_MAX_DEPTH`, on the number of elements that may be submitted to FindStat, which is currently 200. Therefore, the interface tries to truncate queries appropriately, but this may be impossible, especially with distribution searches:

```
sage: S6 = Permutations(6); S6.cardinality()                              # optional -- internet
720
sage: findstat((S6, [1 for a in S6]))                                     # optional -- internet
Traceback (most recent call last):
...
ValueError: After discarding elements not in the range, too few (=0) values remained to send to
```

browse()

Open the FindStat web page in a browser.

EXAMPLES:

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

login()

Open the FindStat login page in a browser.

EXAMPLES:

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

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

Set the user for this 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")
```

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

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

Bases: `sage.structure.element.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
Cc0005: Dyck paths
```

SEEALSO:

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
```

first_terms(*statistic*, *max_values=1200*)

Compute the first few terms of the given statistic.

INPUT:

- statistic – a callable.
- max_values – the number of terms to compute at most.

OUTPUT:

A 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, max_values=10)                # optional -- internet,r
[([[0]], 1),
 ([[1]], 1),
 ([[2]], 1),
 ([[3]], 1),
 ([[0, 0], [0]], 1),
 ([[1, 0], [0]], 1),
 ([[1, 0], [1]], 1),
 ([[1, 1], [1]], 1),
 ([[2, 0], [0]], 1),
 ([[2, 0], [1]], 1)]
```

from_string()

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

OUTPUT:

The function that produces the sage object given its FindStat normal 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._sageconstructor(6)                                # 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
```

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'
```

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

```
sage: c.in_range(GelfandTsetlinPattern([[3, 1], [1]]))                # optional -- internet
True
sage: c.in_range(GelfandTsetlinPattern([[4, 1], [1]]))                # optional -- internet,r
False
```

TESTS:

```
sage: from sage.databases.findstat import FindStatCollections
sage: l = FindStatCollections()                                       # optional -- internet
sage: long = [9, 12, 14, 20]
sage: for c in l:                                                     # optional -- internet,
....:     if c.id() not in long and c.is_supported():
....:         f = c.first_terms(lambda x: 1, max_values=10000)
....:         print c, len(f), all(c.in_range(e) for e, _ in f)
....:
Cc0001: Permutations 10000 True
Cc0002: Integer partitions 270 True
Cc0005: Dyck paths 2054 True
Cc0006: Integer compositions 510 True
Cc0007: Standard tableaux 3734 True
Cc0010: Binary trees 2054 True
Cc0013: Cores 100 True
Cc0017: Alternating sign matrices 7916 True
Cc0018: Gelfand-Tsetlin patterns 934 True
Cc0019: Semistandard tableaux 10000 True
Cc0021: Ordered trees 2055 True
Cc0022: Finite Cartan types 31 True
Cc0023: Parking functions 10000 True
```

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,
False
```

name()

Return the name of the FindStat collection.

OUTPUT:

The name of the FindStat collection, in singular.

EXAMPLES:

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

to_string()

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

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, 2), (2, 1)], 3)'
```

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

Bases: sage.structure.parent.Parent, sage.structure.unique_representation.UniqueRepresenta

The class of FindStat collections.

The elements of this class are combinatorial collections in FindStat as of August 2015. If a new collection was added to the web service since then, the dictionary _findstat_collections in this class has to be updated accordingly.

EXAMPLES:

```
sage: from sage.databases.findstat import FindStatCollections
sage: sorted(c for c in FindStatCollections())                   # 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]
```

Element

alias of FindStatCollection

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

Bases: sage.structure.element.Element

A FindStat map.

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

The result of a findstat query contains a (possibly empty) list of such maps. This class provides methods to inspect various properties of these maps, in particular code().

INPUT:

- a string containing the FindStat name of the map, or an integer representing its FindStat id.

EXAMPLES:

```
sage: from sage.databases.findstat import FindStatMap
sage: FindStatMap(71)                                            # optional -- internet
Mp00071: descent composition
sage: FindStatMap("descent composition")                         # optional -- internet
Mp00071: descent composition
```

SEEALSO:

FindStatMaps

code()

Return the code associated with the map.

OUTPUT:

A string.

EXAMPLES:

```
sage: from sage.databases.findstat import FindStatMap              # optional -- internet
sage: print FindStatMap(71).code()                                 # optional -- internet
def descents_composition(elt):
    if len(elt) == 0:
        return Composition([])
    d = [-1] + elt.descents() + [len(elt)-1]
    return Composition([ d[i+1]-d[i] for i in range(len(d)-1)])
```

code_name()

Return the name of the function defined by code().

OUTPUT:

A string.

EXAMPLES:

```
sage: from sage.databases.findstat import FindStatMap              # optional -- internet
sage: print FindStatMap(71).code_name()                            # optional -- internet
descents_composition
```

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(71).codomain()                                   # optional -- internet
Cc0006: Integer compositions
```

description()

Return the FindStat description of the map.

OUTPUT:

The description as a string.

EXAMPLES:

```
sage: m = findstat("Permutations", lambda pi: pi.length())[1][1][0] # optional -- internet
sage: print m.description()                                        # optional -- internet,r
Let $\sigma \in \mathcal{S}_n$ be a permutation.

Maps $\sigma$ to the permutation $\tau$ such that the major code of $\tau$ is given by the I

In particular, the number of inversions of $\sigma$ equals the major index of $\tau$.
```

```
EXAMPLES:

$[3,4,1,2] \mapsto [3,1,4,2]$
```

domain()

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

OUTPUT:

The domain of the map as a `FindStatCollection`.

EXAMPLES:

```
sage: from sage.databases.findstat import FindStatMap                 # optional -- internet
sage: FindStatMap(71).domain()                                         # optional -- internet
Cc0001: Permutations
```

id()

Return the FindStat identifier of the map.

OUTPUT:

The FindStat identifier of the map as an integer.

EXAMPLES:

```
sage: m = findstat("Permutations", lambda pi: pi.length())[1][1][0] # optional -- internet
sage: m.id()                                                           # optional -- internet
62
```

id_str()

Return the FindStat identifier of the map.

OUTPUT:

The FindStat identifier of the map as a string.

EXAMPLES:

```
sage: m = findstat("Permutations", lambda pi: pi.length())[1][1][0] # optional -- internet
sage: m.id_str()                                                       # optional -- internet
'Mp00062'
```

name()

Return the FindStat name of the map.

OUTPUT:

The name of the map as a string, as used by FindStat.

EXAMPLES:

```
sage: m = findstat("Permutations", lambda pi: pi.length())[1][1][0] # optional -- internet
sage: m.name()                                                         # optional -- internet
u'inversion-number to major-index bijection'
```

class sage.databases.findstat.**FindStatMaps**

Bases: `sage.structure.parent.Parent`, `sage.structure.unique_representation.UniqueRepresenta`

The class of FindStat maps.

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

EXAMPLES:

We can print a nice list of maps currently in FindStat, sorted by domain and codomain:

```
sage: from sage.databases.findstat import FindStatMap, FindStatMaps
sage: for m in sorted(FindStatMaps(), key=lambda m: (m.domain(), m.codomain())):    # optional --
....:     print m.domain().name().ljust(30), m.codomain().name().ljust(30), m.name()
....:
Permutation                    Standard tableau               Robinson-Schensted insertion table
Permutation                    Integer partition              Robinson-Schensted tableau shape
Permutation                    Binary tree                    to increasing tree
...
```

Element
alias of FindStatMap

class sage.databases.findstat.**FindStatStatistic**(*id, first_terms=None, data=None, function=None, code='', collection=None, depth=None*)

Bases: sage.structure.sage_object.SageObject

The class of FindStat statistics.

Do not instantiate this class directly. Instead, use findstat.

browse()

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

EXAMPLES:

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

code()

Return the code associated with the statistic.

OUTPUT:

A 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(1).code()                                       # optional -- internet,r
def statistic(x):
    return len(x.reduced_words())

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

collection()

Return the FindStat collection of the statistic.

OUTPUT:

The FindStat collection of the statistic as an instance of FindStatCollection.

EXAMPLES:

```
sage: findstat(1).collection()                                       # optional -- internet
Cc0001: Permutations
```

data()

Return the data used for querying the FindStat database.

OUTPUT:

The data provided by the user to query the FindStat database. When the database was searched using an identifier, `data` is `None`.

EXAMPLES:

```
sage: S4 = Permutations(4); findstat((S4, [pi.length() for pi in S4])).data() # optional --
[(Standard permutations of 4,
  ['[1, 2, 3, 4]',
   '[1, 2, 4, 3]',
   '[1, 3, 2, 4]',
   '[1, 3, 4, 2]',
   '[1, 4, 2, 3]',
   '[1, 4, 3, 2]',
   '[2, 1, 3, 4]',
   '[2, 1, 4, 3]',
   '[2, 3, 1, 4]',
   '[2, 3, 4, 1]',
   '[2, 4, 1, 3]',
   '[2, 4, 3, 1]',
   '[3, 1, 2, 4]',
   '[3, 1, 4, 2]',
   '[3, 2, 1, 4]',
   '[3, 2, 4, 1]',
   '[3, 4, 1, 2]',
   '[3, 4, 2, 1]',
   '[4, 1, 2, 3]',
   '[4, 1, 3, 2]',
   '[4, 2, 1, 3]',
   '[4, 2, 3, 1]',
   '[4, 3, 1, 2]',
   '[4, 3, 2, 1]'],
  [0, 1, 1, 2, 2, 3, 1, 2, 2, 3, 3, 4, 2, 3, 3, 4, 4, 5, 3, 4, 4, 5, 5, 6])]
```

description()

Return the description of the statistic.

OUTPUT:

A string, whose first line is used as the name of the statistic.

EXAMPLES:

```
sage: print findstat(1).description()                                # optional -- internet,r
The number of ways to write a permutation as a minimal length product of simple transpositio

That is, the number of reduced words for the permutation.  E.g., there are two reduced words
```

edit(*max_values=1200*)

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

INPUT:

- `max_values` – integer (default: `FINDSTAT_MAX_SUBMISSION_VALUES`); if `function()` is defined and the statistic is a new statistic, use `FindStatCollection.first_terms()` to produce at most `max_values` terms.

OUTPUT:

•Raise an error if the query has a match with no intermediate combinatorial maps.

EXAMPLES:

```
sage: s = findstat(DyckWords(4), lambda x: randint(1,1000)); s      # optional -- internet
a new statistic on Cc0005: Dyck paths
```

The following uses `lambda x: randint(1,1000)` to produce 14 terms, because `min(DyckWords(4).cardinality(), FINDSTAT_MAX_SUBMISSION_VALUES)` is 14:

```
sage: s.submit()                                                    # optional -- webbrowser
```

first_terms()

Return the first terms of the statistic.

OUTPUT:

A 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. If the statistic is in the FindStat database, the list contains exactly the pairs in the database.

EXAMPLES:

```
sage: findstat(1).first_terms()                                     # optional -- internet,r
[([1], 1),
 ([1, 2], 1),
 ([2, 1], 1),
 ([1, 2, 3], 1),
 ([1, 3, 2], 1),
 ([2, 1, 3], 1),
 ...
```

TESTS:

```
sage: r = findstat({d: randint(1,1000) for d in DyckWords(4)}); r   # optional -- internet
a new statistic on Cc0005: Dyck paths

sage: isinstance(r.first_terms(), list)                             # optional -- internet
True
sage: all(isinstance(e, tuple) and len(e)==2 and isinstance(e[1], (ZZ, Integer, int)) for e
True
```

first_terms_str()

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: findstat(1).first_terms_str()[:10]                            # optional -- internet,r
'[1] => 1\r\n'
```

function()

Return the function used to compute the values of the statistic.

OUTPUT:

The function used to compute the values of the statistic, or `None`.

EXAMPLES:

```
sage: findstat("Permutations", lambda pi: pi.length()).function()    # optional -- internet
...
<function <lambda> at ...>
```

generating_functions(*style='polynomial'*)

Return the generating functions of `self` in a dictionary.

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

INPUT:

- a 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.

EXAMPLES:

```
sage: st = findstat(18)                                            # optional -- internet

sage: st.generating_functions()                                    # optional -- internet,r
{2: q + 1,
 3: q^3 + 2*q^2 + 2*q + 1,
 4: q^6 + 3*q^5 + 5*q^4 + 6*q^3 + 5*q^2 + 3*q + 1,
 5: q^10 + 4*q^9 + 9*q^8 + 15*q^7 + 20*q^6 + 22*q^5 + 20*q^4 + 15*q^3 + 9*q^2 + 4*q + 1,
 6: q^15 + 5*q^14 + 14*q^13 + 29*q^12 + 49*q^11 + 71*q^10 + 90*q^9 + 101*q^8 + 101*q^7 + 90*

sage: st.generating_functions(style="dictionary")                  # optional -- internet,r
{2: {0: 1, 1: 1},
 3: {0: 1, 1: 2, 2: 2, 3: 1},
 4: {0: 1, 1: 3, 2: 5, 3: 6, 4: 5, 5: 3, 6: 1},
 5: {0: 1, 1: 4, 2: 9, 3: 15, 4: 20, 5: 22, 6: 20, 7: 15, 8: 9, 9: 4, 10: 1},
 6: {0: 1,
  1: 5,
  2: 14,
  3: 29,
  4: 49,
  5: 71,
  6: 90,
  7: 101,
  8: 101,
  9: 90,
  10: 71,
  11: 49,
  12: 29,
  13: 14,
  14: 5,
  15: 1}}

sage: st.generating_functions(style="list")                        # optional -- internet,r
{2: [1, 1],
 3: [1, 2, 2, 1],
```

```
4: [1, 3, 5, 6, 5, 3, 1],
5: [1, 4, 9, 15, 20, 22, 20, 15, 9, 4, 1],
6: [1, 5, 14, 29, 49, 71, 90, 101, 101, 90, 71, 49, 29, 14, 5, 1]}
```

id()

Return the FindStat identifier of the statistic.

OUTPUT:

The FindStat identifier of the statistic (or 0), as an integer.

EXAMPLES:

```
sage: findstat(1).id()                                    # optional -- internet
1
```

id_str()

Return the FindStat identifier of the statistic.

OUTPUT:

The FindStat identifier of the statistic (or 'St000000'), as a string.

EXAMPLES:

```
sage: findstat(1).id_str()                                # optional -- internet
'St000001'
```

modified()

Return whether the statistic was modified.

OUTPUT:

True, if the statistic was modified using `set_description()`, `set_code()`, `set_references()`, etc. False otherwise.

EXAMPLES:

```
sage: findstat(41).set_description("")                    # optional -- internet
sage: findstat(41).modified()                             # optional -- internet
True
```

name()

Return the name of the statistic.

OUTPUT:

A string, which is just the first line of the description of the statistic.

EXAMPLES:

```
sage: findstat(1).name()                                  # optional -- internet,r
u'The number of ways to write a permutation as a minimal length product of simple transposit
```

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 too big, the OEIS result is corrupted.
- `verbose` (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(18)                                    # optional -- internet

sage: st.oeis_search()                                     # optional -- internet,r
Searching the OEIS for "1,1  1,2,2,1  1,3,5,6,5,3,1  1,4,9,15,20,22,20,15,9,4,1  1,5,14,29,4

0: A008302: Triangle of Mahonian numbers T(n,k): coefficients in expansion of Product_{i=0..

sage: st.oeis_search(search_size=13)                       # optional -- internet,r
Searching the OEIS for "1,1  1,2,2,1  1,3,5,6,5,3,1"

0: A008302: Triangle of Mahonian numbers T(n,k): coefficients in expansion of Product_{i=0..
1: A115570: Array read by rows: row n (n>= 1) gives the Betti numbers for the n-th element c
2: A187447: Array for all multiset choices (multiset repetition class representatives in Abr
```

references()

Return the references associated with the statistic.

OUTPUT:

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

Todo

Since the references in the database are sometimes not formatted properly, this method is unreliable. The string representation can be obtained via `_references`.

EXAMPLES:

```
sage: findstat(1).references()                             # optional -- internet,r
0: P. Edelman and C. Greene, Balanced tableaux, Adv. in Math., 63 (1987), pp. 42-99.
1: [[OEIS:A005118]]
2: [[oeis:A246865]]
```

set_code(*value*)

Set the code associated with the statistic.

INPUT:

- a string – contributors are encouraged to submit sage code in the form:

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

OUTPUT:

- Raise an error if the query has a match with no intermediate combinatorial maps.

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

EXAMPLES:

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

set_description (*value*)

Set the description of the statistic.

INPUT:

- a string – the name of the statistic followed by its description on a separate line.

OUTPUT:

- Raise an error, if the query has a match with no intermediate combinatorial maps.

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

EXAMPLES:

```
sage: s = findstat([(d, randint(1,1000)) for d in DyckWords(4)]); s # optional -- internet
a new statistic on Cc0005: Dyck paths
sage: s.set_description("Random values on Dyck paths.\r\nNot for submssion.") # optional --
sage: s                                                             # optional -- internet
a new statistic on Cc0005: Dyck paths
sage: s.name()                                                      # optional -- internet
'Random values on Dyck paths.'
sage: print s.description()                                         # optional -- internet
Random values on Dyck paths.
Not for submssion.
```

set_references (*value*)

Set the references associated with the statistic.

INPUT:

- a string – the individual references should be separated by FINDSTAT_SEPARATOR_REFERENCES, which is "\r\n".

OUTPUT:

- Raise an error, if the query has a match with no intermediate combinatorial maps.

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

EXAMPLES:

```
sage: s = findstat([(d, randint(1,1000)) for d in DyckWords(4)]); s # optional -- internet
a new statistic on Cc0005: Dyck paths
sage: s.set_references("[1] The wonders of random Dyck paths, Anonymous Coward, [[arXiv:1102
sage: s.references()                                                # optional -- internet
0: [1] The wonders of random Dyck paths, Anonymous Coward, [[arXiv:1102.4226]].
1: [2] [[oeis:A000001]]
```

submit (*max_values=1200*)

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

INPUT:

- `max_values` – integer (default: `FINDSTAT_MAX_SUBMISSION_VALUES`); if `function()` is defined and the statistic is a new statistic, use `FindStatCollection.first_terms()` to produce at most `max_values` terms.

OUTPUT:

- Raise an error if the query has a match with no intermediate combinatorial maps.

EXAMPLES:

```
sage: s = findstat(DyckWords(4), lambda x: randint(1,1000)); s      # optional -- internet
a new statistic on Cc0005: Dyck paths
```

The following uses `lambda x: randint(1,1000)` to produce 14 terms, because `min(DyckWords(4).cardinality(), FINDSTAT_MAX_SUBMISSION_VALUES)` is 14:

```
sage: s.submit()                                              # optional -- webbrowser
```

CHAPTER

SEVEN

FRANK LUEBECK'S TABLES OF CONWAY POLYNOMIALS OVER FINITE FIELDS

class `sage.databases.conway.ConwayPolynomials`

Bases: `_abcoll.Mapping`

Initialize the database.

TESTS:

```
sage: c = ConwayPolynomials()
sage: c
Frank Luebeck's database of Conway polynomials
```

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))
[]
```

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
sage: c.has_polynomial(60821, 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, 0, 1)
sage: c.polynomial(97, 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
```

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

Bases: _abcoll.Mapping

Places dict into a non-mutable mapping.

TESTS:

```
sage: from sage.databases.conway import DictInMapping
sage: d = {}
sage: m = DictInMapping(d); m
{}
sage: d[0] = 1; m
{0: 1}
sage: m[2] = 3
Traceback (most recent call last):
...
TypeError: 'DictInMapping' object does not support item assignment
```

CHAPTER

EIGHT

TABLES OF ZEROS OF THE RIEMANN-ZETA FUNCTION

AUTHORS:

- William Stein: initial version
- Jeroen Demeyer (2015-01-20): convert `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.

In order to use `zeta_zeros()`, you will need to install the optional Odlyzko database package:

```
sage -i database_odlyzko_zeta
```

You can see a list of all available optional packages with `sage --optional`.

REFERENCES:

- http://www.dtc.umn.edu/~odlyzko/zeta_tables/index.html

EXAMPLES:

The following example prints the imaginary part of the 13th nontrivial zero of the Riemann zeta function:

```
sage: zz = zeta_zeros()   # optional - database_odlyzko_zeta
sage: zz[12]              # optional - database_odlyzko_zeta
59.347044003
sage: len(zz)             # optional - database_odlyzko_zeta
2001052
```

CHAPTER NINE

IDEALS FROM THE SYMBOLIC DATA PROJECT

This file implements a thin wrapper for the optional symbolic data set 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.

EXAMPLES:

```
sage: sd = SymbolicData(); sd # optional - database_symbolic_data
SymbolicData with 372 ideals

sage: sd.ZeroDim__example_1 # optional - database_symbolic_data
Ideal (x1^2 + x2^2 - 10, x1^2 + x1*x2 + 2*x2^2 - 16) of Multivariate Polynomial Ring in x1, x2 over F

sage: sd.Katsura_3 # 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 F

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 F
```

AUTHORS:

- Martin Albrecht <martinralbrecht@googlemail.com>

class `sage.databases.symbolic_data.SymbolicData`

Database of ideals as distributed by the 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'*)

Returns 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_symbclic_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,
```

`trait_names`()

EXAMPLES:

```
sage: sd = SymbolicData() # optional - database_symbolic_data
sage: sorted(sd.trait_names())[:10] # optional - database_symbolic_data
['Bjoerk_8',
 'Bronstein-86',
 'Buchberger-87',
 'Butcher',
 'Caprasse',
 'Cassou',
 'Cohn_2',
 'Curves__curve10_20',
 'Curves__curve10_20',
 'Curves__curve10_30']
```

CHAPTER

TEN

CUNNINGHAM TABLE

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

List of all the prime numbers occuring in the so called Cunningham table. They occur in the factorization of numbers of type $b^n + 1$ or $b^n - 1$ with $b \in \{2, 3, 5, 6, 7, 10, 11, 12\}$. Data from http://cage.ugent.be/~jdemeyer/cunningham/

CHAPTER

ELEVEN

DATABASE OF HILBERT POLYNOMIALS

class sage.databases.db_class_polynomials.**AtkinClassPolynomialDatabase**

Bases: sage.databases.db_class_polynomials.ClassPolynomialDatabase

The database of Atkin class polynomials.

class sage.databases.db_class_polynomials.**ClassPolynomialDatabase**

class sage.databases.db_class_polynomials.**DedekindEtaClassPolynomialDatabase**

Bases: sage.databases.db_class_polynomials.ClassPolynomialDatabase

The database of Dedekind eta class polynomials.

class sage.databases.db_class_polynomials.**HilbertClassPolynomialDatabase**

Bases: sage.databases.db_class_polynomials.ClassPolynomialDatabase

The database of Hilbert class polynomials.

EXAMPLES:

```
sage: db = HilbertClassPolynomialDatabase()
sage: db[-4]                       # optional - database_kohel
x - 1728
sage: db[-7]                       # optional - database_kohel
x + 3375
sage: f = db[-23]; f               # optional - database_kohel
x^3 + 3491750*x^2 - 5151296875*x + 12771880859375
sage: f.discriminant().factor()    # optional - database_kohel
-1 * 5^18 * 7^12 * 11^4 * 17^2 * 19^2 * 23
sage: db[-23]                      # optional - database_kohel
x^3 + 3491750*x^2 - 5151296875*x + 12771880859375
```

class sage.databases.db_class_polynomials.**WeberClassPolynomialDatabase**

Bases: sage.databases.db_class_polynomials.ClassPolynomialDatabase

The database of Weber class polynomials.

CHAPTER

TWELVE

DATABASE OF MODULAR POLYNOMIALS

class `sage.databases.db_modular_polynomials.`**`AtkinModularCorrespondenceDatabase`**

Bases: `sage.databases.db_modular_polynomials.ModularCorrespondenceDatabase`

Initialize the database.

class `sage.databases.db_modular_polynomials.`**`AtkinModularPolynomialDatabase`**

Bases: `sage.databases.db_modular_polynomials.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.

class `sage.databases.db_modular_polynomials.`**`ClassicalModularPolynomialDatabase`**

Bases: `sage.databases.db_modular_polynomials.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).

class `sage.databases.db_modular_polynomials.`**`DedekindEtaModularCorrespondenceDatabase`**

Bases: `sage.databases.db_modular_polynomials.ModularCorrespondenceDatabase`

The database of modular correspondences in $X_0(p) imes X_0(p)$, where the model of the curves $X_0(p) = \mathbf{P}^1$ are specified by quotients of Dedekind's eta function.

class `sage.databases.db_modular_polynomials.`**`DedekindEtaModularPolynomialDatabase`**

Bases: `sage.databases.db_modular_polynomials.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).

class `sage.databases.db_modular_polynomials.`**`ModularCorrespondenceDatabase`**

Bases: `sage.databases.db_modular_polynomials.ModularPolynomialDatabase`

class `sage.databases.db_modular_polynomials.`**`ModularPolynomialDatabase`**

CHAPTER

THIRTEEN

INDICES AND TABLES

- Index
- Module Index
- Search Page

[SteinWatkins] William Stein and Mark Watkins, *A database of elliptic curves—first report*. In *Algorithmic number theory (ANTS V), Sydney, 2002*, Lecture Notes in Computer Science 2369, Springer, 2002, p267–275. http://modular.math.washington.edu/papers/stein-watkins/

PYTHON MODULE INDEX

d

sage.databases.conway, 69
sage.databases.cremona, 3
sage.databases.cunningham_tables, 75
sage.databases.db_class_polynomials, 77
sage.databases.db_modular_polynomials, 79
sage.databases.findstat, 49
sage.databases.jones, 21
sage.databases.odlyzko, 71
sage.databases.oeis, 25
sage.databases.sloane, 45
sage.databases.stein_watkins, 17
sage.databases.symbolic_data, 73

Symbols

__call__() (sage.databases.oeis.OEISSequence method), 31

A

allbsd() (sage.databases.cremona.LargeCremonaDatabase method), 4
allcurves() (sage.databases.cremona.MiniCremonaDatabase method), 5
allgens() (sage.databases.cremona.LargeCremonaDatabase method), 4
AtkinClassPolynomialDatabase (class in sage.databases.db_class_polynomials), 77
AtkinModularCorrespondenceDatabase (class in sage.databases.db_modular_polynomials), 79
AtkinModularPolynomialDatabase (class in sage.databases.db_modular_polynomials), 79
author() (sage.databases.oeis.OEISSequence method), 32

B

browse() (sage.databases.findstat.FindStat method), 53
browse() (sage.databases.findstat.FindStatCollection method), 54
browse() (sage.databases.findstat.FindStatStatistic method), 60
browse() (sage.databases.oeis.OEIS method), 29
browse() (sage.databases.oeis.OEISSequence method), 32
build() (in module sage.databases.cremona), 11

C

class_to_int() (in module sage.databases.cremona), 11
ClassicalModularPolynomialDatabase (class in sage.databases.db_modular_polynomials), 79
ClassPolynomialDatabase (class in sage.databases.db_class_polynomials), 77
cmp_code() (in module sage.databases.cremona), 11
code() (sage.databases.findstat.FindStatMap method), 58
code() (sage.databases.findstat.FindStatStatistic method), 60
code_name() (sage.databases.findstat.FindStatMap method), 58
codomain() (sage.databases.findstat.FindStatMap method), 58
coefficients_and_data() (sage.databases.cremona.MiniCremonaDatabase method), 5
collection() (sage.databases.findstat.FindStatStatistic method), 60
comments() (sage.databases.oeis.OEISSequence method), 32
conductor_range() (sage.databases.cremona.MiniCremonaDatabase method), 6
ConwayPolynomials (class in sage.databases.conway), 69
copy_gz_file() (in module sage.databases.sloane), 46
cremona_letter_code() (in module sage.databases.cremona), 11
cremona_to_lmfdb() (in module sage.databases.cremona), 12

CremonaDatabase() (in module sage.databases.cremona), 3
cross_references() (sage.databases.oeis.OEISSequence method), 33
cunningham_prime_factors() (in module sage.databases.cunningham_tables), 75
curves() (sage.databases.cremona.MiniCremonaDatabase method), 6

D

data() (sage.databases.findstat.FindStatStatistic method), 60
data_from_coefficients() (sage.databases.cremona.MiniCremonaDatabase method), 6
DedekindEtaClassPolynomialDatabase (class in sage.databases.db_class_polynomials), 77
DedekindEtaModularCorrespondenceDatabase (class in sage.databases.db_modular_polynomials), 79
DedekindEtaModularPolynomialDatabase (class in sage.databases.db_modular_polynomials), 79
degphi() (sage.databases.cremona.LargeCremonaDatabase method), 5
degrees() (sage.databases.conway.ConwayPolynomials method), 69
description() (sage.databases.findstat.FindStatMap method), 58
description() (sage.databases.findstat.FindStatStatistic method), 61
DictInMapping (class in sage.databases.conway), 70
domain() (sage.databases.findstat.FindStatMap method), 59

E

ecdb_num_curves() (in module sage.databases.stein_watkins), 19
edit() (sage.databases.findstat.FindStatStatistic method), 61
Element (sage.databases.findstat.FindStatCollections attribute), 57
Element (sage.databases.findstat.FindStatMaps attribute), 60
elliptic_curve() (sage.databases.cremona.MiniCremonaDatabase method), 7
elliptic_curve_from_ainvs() (sage.databases.cremona.MiniCremonaDatabase method), 7
examples() (sage.databases.oeis.OEISSequence method), 33
extensions_or_errors() (sage.databases.oeis.OEISSequence method), 34

F

FancyTuple (class in sage.databases.oeis), 27
find() (sage.databases.sloane.SloaneEncyclopediaClass method), 46
find_by_description() (sage.databases.oeis.OEIS method), 29
find_by_id() (sage.databases.oeis.OEIS method), 30
find_by_subsequence() (sage.databases.oeis.OEIS method), 30
FindStat (class in sage.databases.findstat), 51
FindStatCollection (class in sage.databases.findstat), 53
FindStatCollections (class in sage.databases.findstat), 57
FindStatMap (class in sage.databases.findstat), 57
FindStatMaps (class in sage.databases.findstat), 59
FindStatStatistic (class in sage.databases.findstat), 60
first_terms() (sage.databases.findstat.FindStatCollection method), 54
first_terms() (sage.databases.findstat.FindStatStatistic method), 62
first_terms() (sage.databases.oeis.OEISSequence method), 34
first_terms_str() (sage.databases.findstat.FindStatStatistic method), 62
formulas() (sage.databases.oeis.OEISSequence method), 35
from_string() (sage.databases.findstat.FindStatCollection method), 54
function() (sage.databases.findstat.FindStatStatistic method), 62

G

generating_functions() (sage.databases.findstat.FindStatStatistic method), 63

get() (sage.databases.jones.JonesDatabase method), 21
get_ideal() (sage.databases.symbolic_data.SymbolicData method), 74

H

has_polynomial() (sage.databases.conway.ConwayPolynomials method), 69
HilbertClassPolynomialDatabase (class in sage.databases.db_class_polynomials), 77

I

id() (sage.databases.findstat.FindStatCollection method), 55
id() (sage.databases.findstat.FindStatMap method), 59
id() (sage.databases.findstat.FindStatStatistic method), 64
id() (sage.databases.oeis.OEISSequence method), 35
id_str() (sage.databases.findstat.FindStatCollection method), 55
id_str() (sage.databases.findstat.FindStatMap method), 59
id_str() (sage.databases.findstat.FindStatStatistic method), 64
in_range() (sage.databases.findstat.FindStatCollection method), 55
install() (sage.databases.sloane.SloaneEncyclopediaClass method), 46
install_from_gz() (sage.databases.sloane.SloaneEncyclopediaClass method), 46
is_finite() (sage.databases.oeis.OEISSequence method), 35
is_full() (sage.databases.oeis.OEISSequence method), 36
is_optimal_id() (in module sage.databases.cremona), 13
is_supported() (sage.databases.findstat.FindStatCollection method), 56
isogeny_class() (sage.databases.cremona.MiniCremonaDatabase method), 8
isogeny_classes() (sage.databases.cremona.MiniCremonaDatabase method), 8
iter() (sage.databases.cremona.MiniCremonaDatabase method), 8
iter_levels() (sage.databases.stein_watkins.SteinWatkinsAllData method), 18
iter_optimal() (sage.databases.cremona.MiniCremonaDatabase method), 9

J

JonesDatabase (class in sage.databases.jones), 21

K

keywords() (sage.databases.oeis.OEISSequence method), 37

L

LargeCremonaDatabase (class in sage.databases.cremona), 4
largest_conductor() (sage.databases.cremona.MiniCremonaDatabase method), 9
links() (sage.databases.oeis.OEISSequence method), 37
list() (sage.databases.cremona.MiniCremonaDatabase method), 9
list_optimal() (sage.databases.cremona.MiniCremonaDatabase method), 9
lmfdb_to_cremona() (in module sage.databases.cremona), 13
load() (sage.databases.sloane.SloaneEncyclopediaClass method), 46
login() (sage.databases.findstat.FindStat method), 53

M

MiniCremonaDatabase (class in sage.databases.cremona), 5
modified() (sage.databases.findstat.FindStatStatistic method), 64
ModularCorrespondenceDatabase (class in sage.databases.db_modular_polynomials), 79
ModularPolynomialDatabase (class in sage.databases.db_modular_polynomials), 79

N

name() (sage.databases.findstat.FindStatCollection method), 56
name() (sage.databases.findstat.FindStatMap method), 59
name() (sage.databases.findstat.FindStatStatistic method), 64
name() (sage.databases.oeis.OEISSequence method), 38
natural_object() (sage.databases.oeis.OEISSequence method), 38
next() (sage.databases.stein_watkins.SteinWatkinsAllData method), 19
number_of_curves() (sage.databases.cremona.MiniCremonaDatabase method), 10
number_of_isogeny_classes() (sage.databases.cremona.MiniCremonaDatabase method), 10

O

OEIS (class in sage.databases.oeis), 27
oeis_search() (sage.databases.findstat.FindStatStatistic method), 64
OEISSequence (class in sage.databases.oeis), 30
offsets() (sage.databases.oeis.OEISSequence method), 40
old_cremona_letter_code() (in module sage.databases.cremona), 13
old_IDs() (sage.databases.oeis.OEISSequence method), 40

P

parse_cremona_label() (in module sage.databases.cremona), 14
parse_lmfdb_label() (in module sage.databases.cremona), 15
parse_sequence() (in module sage.databases.sloane), 47
polynomial() (sage.databases.conway.ConwayPolynomials method), 69
primes() (sage.databases.conway.ConwayPolynomials method), 70
programs() (sage.databases.oeis.OEISSequence method), 40

R

ramified_at() (sage.databases.jones.JonesDatabase method), 22
random() (sage.databases.cremona.MiniCremonaDatabase method), 10
raw_entry() (sage.databases.oeis.OEISSequence method), 41
references() (sage.databases.findstat.FindStatStatistic method), 65
references() (sage.databases.oeis.OEISSequence method), 41

S

sage.databases.conway (module), 69
sage.databases.cremona (module), 3
sage.databases.cunningham_tables (module), 75
sage.databases.db_class_polynomials (module), 77
sage.databases.db_modular_polynomials (module), 79
sage.databases.findstat (module), 49
sage.databases.jones (module), 21
sage.databases.odlyzko (module), 71
sage.databases.oeis (module), 25
sage.databases.sloane (module), 45
sage.databases.stein_watkins (module), 17
sage.databases.symbolic_data (module), 73
sequence_name() (sage.databases.sloane.SloaneEncyclopediaClass method), 46
set_code() (sage.databases.findstat.FindStatStatistic method), 65
set_description() (sage.databases.findstat.FindStatStatistic method), 65

set_references() (sage.databases.findstat.FindStatStatistic method), 66
set_user() (sage.databases.findstat.FindStat method), 53
show() (sage.databases.oeis.OEISSequence method), 42
sloane_find() (in module sage.databases.sloane), 47
sloane_sequence() (in module sage.databases.sloane), 47
SloaneEncyclopediaClass (class in sage.databases.sloane), 46
smallest_conductor() (sage.databases.cremona.MiniCremonaDatabase method), 10
split_code() (in module sage.databases.cremona), 16
SteinWatkinsAllData (class in sage.databases.stein_watkins), 18
SteinWatkinsIsogenyClass (class in sage.databases.stein_watkins), 19
SteinWatkinsPrimeData (class in sage.databases.stein_watkins), 19
submit() (sage.databases.findstat.FindStatStatistic method), 66
SymbolicData (class in sage.databases.symbolic_data), 73

T

to_string() (sage.databases.findstat.FindStatCollection method), 56
to_tuple() (in module sage.databases.oeis), 43
trait_names() (sage.databases.symbolic_data.SymbolicData method), 74

U

unload() (sage.databases.sloane.SloaneEncyclopediaClass method), 46
unramified_outside() (sage.databases.jones.JonesDatabase method), 22
url() (sage.databases.oeis.OEISSequence method), 43

W

WeberClassPolynomialDatabase (class in sage.databases.db_class_polynomials), 77

Z

zeta_zeros() (in module sage.databases.odlyzko), 71