sudokutools Documentation

Release 0.5.0-dev

Maik Messerschmidt

Apr 12, 2020

Contents

Yet another python sudoku library.

CHAPTER 1

README

You can find a short introduction on the project site.

CHAPTER 2

Contents

2.1 The sudokutools library

Yet another python sudoku library.

sudokutools is a collection of functions and classes, which enable you to read, create, analyze, solve and print sudokus.

For a short introduction see: https://github.com/messersm/sudokutools

Package modules:

- sudokutools.analyze: Check, rate and analyze sudokus.
- sudokutools.dlx: Internal module - do not use.
- sudokutools.generate: Create new sudokus.
- sudokutools.solve: Low-level solving of sudokus.
- sudokutools.solvers: High level solving of sudokus.
- sudokutools.sudoku: Parse, print and compare sudokus.

2.1.1 `sudokutools.sudoku` - Parsing, printing and comparing

Parse, print and compare sudokus.

Classes defined here:

- Sudoku: Represents a sudoku.

Functions defined here:

- view(): Return sudoku (with candidates) as a human-readable string.

class `sudokutools.sudoku.`**Sudoku**(*box_size=(3, 3)*)

Bases: `object`

Represents a sudoku.

This class provides methods to read, write, copy and display data to and from a sudoku as well as compare one sudoku with another. It stores 9x9 fields and the numbers and candidates within these fields.

Coordinates for row and column access are values from 0 to 8 (including). Using other values will most likely raise an IndexError (even though negative numbers are supported, they should not be used).

Overview:

Data access (read, write):

- __getitem__()
- __setitem__()
- get_candidates()
- set_candidates()
- remove_candidates()

Copying:

- copy()

Comparing:

- empty()
- __len__()
- __eq__()
- diff()

Printing:

- __str__()
- encode()

Parsing:

- decode()

__eq__(*other*)
Return if other is equal in all fields.

Parameters **other** (Sudoku) – Most likely another *Sudoku* instance, but any object, which can be accessed by other[row, col] is valid.

Returns

True, if all fields are equal and false if not or other is an incompatible type.

Return type bool

__getitem__(*key*)
Return the number in the field referenced by key.

Parameters **key** (*int, int*) – row and column of the requested field. Must be in range(0, width * height).

Returns The number in the given field, 0 representing an empty field.

Return type int

Raises IndexError – if the given coordinates are not valid.

__init__(*box_size=(3, 3)*)
Create a new empty sudoku.

Parameters **box_size** (*int, int*) – box_width, box_height for the new sudoku. A standard 9x9 sudoku has size=(3, 3).

__iter__()
Iterate through all coordinates of the sudoku.

Yields: (int, int): row and column of each field.

__len__()
Return the number of fields in this sudoku.

Returns The number of fields in this sudoku.

Return type int

__setitem__(*key*, *value*)
Set the number in the field referenced by key to value.

Parameters

- **key** (*int, int*) – row and column of the requested field. Must be in range(0, width * height).
- **value** (*int*) – The number to set the field to, 0 representing an empty field.

Raises IndexError – if the given coordinates are not valid.

__str__()
Return sudoku as a human-readable string.

Returns String representing the sudoku.

Return type str

_quad_without_row_and_column_of(*row*, *col*)
Return some coordinates in the square of (col, row) as a list.

The coordinates in the same row and column are removed.

This is an internal function and should not be used outside of the sudoku module.

box_at(*row*, *col*)
Return the box index of the field at (row, col)

Parameters

- **row** (*int*) – The row of the field.
- **col** (*int*) – The column of the field.

Returns The index of the box, in which the field (row, col) lies.

Return type int

box_of(*row*, *col*, *include=True*)
Return all coordinates in the region of (col, row) as a list.

Parameters

- **row** (*int*) – The row of the field.
- **col** (*int*) – The column of the field.
- **include** (*bool*) – Whether or not to include (row, col).

Returns

list of pairs (row, column) of all fields in the same box (region).

Return type list of (int, int)

column_of(*row*, *col*, *include=True*)
Return all coordinates in the column of (col, row) as a list.

Parameters

- **row** (*int*) – The row of the field.
- **col** (*int*) – The column of the field.
- **include** (*bool*) – Whether or not to include (row, col).

Returns

list of pairs (row, column) of all fields in the same column.

Return type list of (int, int)

copy(*include_candidates=False*)
Returns a copy of this sudoku.

Parameters **include_candidates** (*bool*) – Whether to copy candidates as well.

Returns The new sudoku instance.

Return type *Sudoku*

count()
Return the number of filled fields.

Returns number of filled fields.

Return type int

classmethod decode(*s*, *empty='0'*, *number_sep=None*, *sudoku_sep='|'*, *candidate_sep=','*, *box_size=None*)
Create a new sudoku from the string s.

Parameters

- **s** (*str*) – A string representing the sudoku (see below).
- **empty** (*char*) – A character representing empty fields.
- **sudoku_sep** (*char*) – A character, which separates field information from candidate information.
- **candidate_sep** (*char*) – A character separating the candidate lists.
- **box_size** (*int, int*) – box_width and box_height of the new sudoku. If not provided this will be calculated automatically, which may lead to wrong results. (It will always work as intended if width == height.)

Returns The newly created sudoku.

Return type *Sudoku*

Examples for s: 000030000005009602008004013020060000703040106000080090210300800306800700000020000
000030000005009602008004013020060000703040106000080090210300800306800700000020000|124,235

Whitespace is ignored while parsing the string, so you can place newlines for better readability.

Each number represents the value of a column. If a row is full, we continue in the next one. So the sudoku above looks like this:

```
      |   3   |
    5 |     9 | 6   2
    8 |     4 |   1 3
------+-------+------
  2   |   6   |
7   3 |   4   | 1   6
      |   8   |   9
------+-------+------
2 1   | 3     | 8
3   6 | 8     | 7
      |   2   |
```

The second string additionally defines candidates. Each set of candidates is separated by ',' so the string above defines the candidates for (0, 0) to be 1, 2 and 4 and for (0, 1) to be 2, 3 and 5

This is the default format, which encode() uses and no other format is supported right now.

diff(*other*)

Iterate through coordinates with different values in other.

Compares each field in other to the corresponding field in self and yields the coordinates, if the number within is different.

Parameters **other** (Sudoku) – Most likely another *Sudoku* instance, but any object, which can be accessed by other[row, col] is valid.

Yields *(int, int)* – row and column of the next different field.

empty()

Iterate through the coordinates of all empty fields.

Yields *(int, int)* – row and column of the next empty field.

encode(*row_sep=''*, *col_sep=''*, *include_candidates=False*)

Return sudoku as a (machine-readable) string.

This method is mainly provided to output sudokus in a machine-readable format, but can be used for creating nicely looking representations as well.

Parameters

- **row_sep** (*str*) – Separator between rows.
- **col_sep** (*str*) – Separator between columns.
- **include_candidates** –

Returns String representing this sudoku.

Return type (str)

For examples of default output string see decode().

equals(*other*, *candidates=False*)

filled()

Iterate through the coordinates of filled fields.

Yields *(int, int)* – row and column of the next filled field.

get_candidates(*row*, *col*)
Return the candidates of the field at (row, col).

Parameters

- **row** (*int*) – The row of the field.
- **col** (*int*) – The column of the field.

Returns The candidates at (row, col).

Return type frozenset

get_number(*row*, *col*)
Same as sudoku[row, col].

remove_candidates(*row*, *col*, *value*)
Remove the given candidates in the field at (row, col).

Ignores candidates, which are not present in the field.

Parameters

- **row** (*int*) – The row of the field.
- **col** (*int*) – The column of the field.
- **value** (*iterable*) – The candidates to remove.

row_of(*row*, *col*, *include=True*)
Return all coordinates in the row of (col, row) as a list.

Parameters

- **row** (*int*) – The row of the field.
- **col** (*int*) – The column of the field.
- **include** (*bool*) – Whether or not to include (row, col).

Returns

list of pairs (row, column) of all fields in the same row.

Return type list of (int, int)

set_candidates(*row*, *col*, *value*)
Set the candidates of the field at (row, col) to value.

Parameters

- **row** (*int*) – The row of the field.
- **col** (*int*) – The column of the field.
- **value** (*iterable*) – The candidates to set the field to.

set_number(*row*, *col*, *value*)
Same as sudoku[row, col] = value.

surrounding_of(*row*, *col*, *include=True*)
Return all surrounding coordinates of (col, row) as a list.

Parameters

- **row** (*int*) – The row of the field.
- **col** (*int*) – The column of the field.

- **include** (*bool*) – Whether or not to include (row, col).

Returns

list of pairs (row, column) of all fields in the same column, row or square.

Return type list of (int, int)

sudokutools.sudoku.**view**(*sudoku*, *include_candidates=True*, *number_sep=None*, *candidate_prefix='*'*, *align_right=True*)

Return sudoku as a human-readable string.

Parameters

- **sudoku** – The sudoku to represent.
- **include_candidates** (*bool*) – include candidates (or not)
- **number_sep** (*str*) – separator for candidates. If set to None, this set to ',', if there are numbers > 9 in the sudoku. Otherwise it will be the empty string.
- **candidate_prefix** (*str*) – A string preceding the candidates. This is used to mark output as candidates (for example to recognize naked singles).
- **align_right** (*bool*) – Align field content to the right (will be left-aligned, if set to False).

Returns String representing the sudoku.

Return type str

Example:

```
>>> from sudokutools.solve import init_candidates
>>> from sudokutools.sudoku import Sudoku, view
>>> sudoku = Sudoku.decode('''
... 003020600
... 900305001
... 001806400
... 008102900
... 700000008
... 006708200
... 002609500
... 800203009
... 005010300''')
>>> init_candidates(sudoku)
>>> print(view(sudoku))
    *45   *4578       3 |     *49       2    *147 |       6   *5789     *57
      9  *24678    *47 |       3     *47       5 |    *78    *278       1
    *25    *257       1 |       8     *79       6 |       4  *23579   *2357
----------------------+-----------------------+----------------------
   *345    *345       8 |       1   *3456       2 |       9  *34567  *34567
      7 *123459    *49 |    *459  *34569      *4 |      *1  *13456       8
  *1345  *13459       6 |       7   *3459       8 |       2   *1345    *345
----------------------+-----------------------+----------------------
   *134   *1347       2 |       6    *478       9 |       5   *1478     *47
      8   *1467    *47 |       2    *457       3 |    *17   *1467       9
    *46   *4679       5 |      *4       1     *47 |       3  *24678   *2467
```

2.1.2 sudokutools.solve - Low level solving

Low-level solving of sudokus.

Functions defined here:

- bruteforce(): Solves a sudoku using brute force.
- dlx(): Solves a sudoku using the dancing links algorithm-X.
- calc_candidates(): Calculates candidates of a field in a sudoku.
- init_candidates(): Sets the candidates for all fields in a sudoku.

Warning: Since the functions in this module work using recursion, solving very large Sudokus will likely To be more precise: Python will raise an RecursionError, if `box_width * box_height >= sys.getrecursionlimit()`.

If you really want to generate Sudokus of this size using sudokutools, you have to increase the recursion limit of Python. See: https://stackoverflow.com/questions/3323001/what-is-the-maximum-recursion-depth-in-python-and-how-to-increase-it

sudokutools.solve.**_do_bruteforce**(*sudoku*)
Solve sudoku _inplace_ and yield it in a solved configuration.

This is an internal function and should not be used outside of the solve module.

sudokutools.solve.**bruteforce**(*sudoku*)
Solve the sudoku using brute force and yield solutions.

Parameters **sudoku** (Sudoku) – The Sudoku instance to solve.

Yields *Sudoku* – A solution of the sudoku.

sudokutools.solve.**calc_candidates**(*sudoku*, *row*, *col*)
Return a set of candidates of the sudoku at (row, col).

Parameters

- **sudoku** (Sudoku) – The Sudoku instance for which the candidates are calculated.
- **row** (*int*) – The row of the field
- **col** (*int*) – The column of the field.

Returns A set of candidates for the field at (row, col)

Return type set

sudokutools.solve.**dlx**(*sudoku*)
Solve the sudoku using the dancing links variant of algorithm-X.

Parameters **sudoku** (Sudoku) – The Sudoku instance to solve.

Yields *Sudoku* – A solution of the sudoku.

sudokutools.solve.**init_candidates**(*sudoku*, *filled_only=False*)
Calculate and set all candidates in the sudoku.

Sets all candidates in the sudoku based on the numbers (and nothing else) in the surrounding fields.

Parameters

- **sudoku** (Sudoku) – The Sudoku instance for which the candidates are calculated.
- **filled_only** (*bool*) – Only set candidate of already set fields. E.g. a field with a value of 2, will get the candidates {2}, but a field without a value will get no candidates.

2.1.3 sudokutools.solvers - High level solving

High level solving of sudokus.

This module provides classes which represent typical sudoku solving steps used by humans. Steps can be found and applied to a given sudoku. But steps can also be printed without applying them, e.g. to inform a user, what steps can be taken to solve the sudoku.

A single solve step may consist of multiple actions, e.g.

- Setting a number at a given field.
- Setting the candidates at a given field.
- Removing some of the candidates at a given field.

Solve steps defined here:

- CalcCandidates
- NakedSingle
- NakedPair
- NakedTriple
- NakedQuad
- NakedQuint
- HiddenSingle
- HiddenPair
- HiddenTriple
- HiddenQuad
- HiddenQuint
- PointingPair
- PointingTriple
- XWing
- Swordfish
- Jellyfish
- Bruteforce

class sudokutools.solvers.**Action**

Bases: sudokutools.solvers.ActionTuple

Named tuple, that represents a single change operation on a sudoku.

Create with: Action(func, row, col, value)

Parameters

- **func** (*callable*) – One of Sudoku.set_number, Sudoku.set_candidates and Sudoku.remove_candidates
- **row** (*int*) – The row of the field, which will be changed.
- **col** (*int*) – The column of the field, which will be changed.
- **value** (*int or iterable*) – The number or candidates to set/remove.

class sudokutools.solvers.**BasicFish**(*clues=()*, *affected=()*, *values=()*)
Bases: *sudokutools.solvers.SolveStep*

build_actions(*sudoku*)

classmethod find(*sudoku*)
Iterates through all possible solve steps of this class.

Parameters **sudoku** (Sudoku) – The sudoku to solve.

Yields *SolveStep* – The next solve step.

n = 2

class sudokutools.solvers.**Bruteforce**(*row*, *col*, *value*)
Bases: *sudokutools.solvers._SingleFieldStep*

Solve the sudoku using brute force.

Bruteforce simply works by trial and error testing each combination of valid candidates in a field until a solution has been found.

classmethod find(*sudoku*)
Iterates through all possible solve steps of this class.

Parameters **sudoku** (Sudoku) – The sudoku to solve.

Yields *SolveStep* – The next solve step.

class sudokutools.solvers.**CalculateCandidates**(*clues=()*, *affected=()*, *values=()*)
Bases: *sudokutools.solvers.SolveStep*

Calculates the candidates of fields.

build_actions(*sudoku*)

classmethod find(*sudoku*)
Iterates through all possible solve steps of this class.

Parameters **sudoku** (Sudoku) – The sudoku to solve.

Yields *SolveStep* – The next solve step.

class sudokutools.solvers.**HiddenPair**(*clues=()*, *affected=()*, *values=()*)
Bases: *sudokutools.solvers.HiddenTuple*

n = 2

class sudokutools.solvers.**HiddenQuad**(*clues=()*, *affected=()*, *values=()*)
Bases: *sudokutools.solvers.HiddenTuple*

n = 4

class sudokutools.solvers.**HiddenQuint**(*clues=()*, *affected=()*, *values=()*)
Bases: *sudokutools.solvers.HiddenTuple*

n = 5

class sudokutools.solvers.**HiddenSingle**(*row*, *col*, *value*)
Bases: *sudokutools.solvers._SingleFieldStep*

Finds hidden singles in a sudoku.

A hidden single is a field containing a candidate which exists in no other fields in the same row, column or box.

The field can be set to this candidate and this candidate can be removed from all fields in the same row, column and box.

classmethod find(*sudoku*)
Iterates through all possible solve steps of this class.

Parameters **sudoku** (Sudoku) – The sudoku to solve.

Yields *SolveStep* – The next solve step.

class sudokutools.solvers.**HiddenTriple**(*clues=()*, *affected=()*, *values=()*)
Bases: *sudokutools.solvers.HiddenTuple*

n = 3

class sudokutools.solvers.**HiddenTuple**(*clues=()*, *affected=()*, *values=()*)
Bases: *sudokutools.solvers.SolveStep*

Finds hidden tuples in a sudoku.

A hidden tuple is a set of n fields in a row, column or box, which (in unison) contain a set of at most n candidates, which are present in no other fields of the same row, column or box.

All other candidates can be removed from these fields.

build_actions(*sudoku*)

classmethod find(*sudoku*)
Iterates through all possible solve steps of this class.

Parameters **sudoku** (Sudoku) – The sudoku to solve.

Yields *SolveStep* – The next solve step.

n = 2

class sudokutools.solvers.**Jellyfish**(*clues=()*, *affected=()*, *values=()*)
Bases: *sudokutools.solvers.BasicFish*

n = 4

class sudokutools.solvers.**NakedPair**(*clues=()*, *affected=()*, *values=()*)
Bases: *sudokutools.solvers.NakedTuple*

n = 2

class sudokutools.solvers.**NakedQuad**(*clues=()*, *affected=()*, *values=()*)
Bases: *sudokutools.solvers.NakedTuple*

n = 4

class sudokutools.solvers.**NakedQuint**(*clues=()*, *affected=()*, *values=()*)
Bases: *sudokutools.solvers.NakedTuple*

n = 5

class sudokutools.solvers.**NakedSingle**(*row*, *col*, *value*)
Bases: *sudokutools.solvers._SingleFieldStep*

Finds naked singles in a sudoku.

A naked single is a field with only one candidate.

The field can be set to this candidate and this candidate can be removed from all fields in the same row, column and box.

classmethod find(*sudoku*)
Iterates through all possible solve steps of this class.

Parameters **sudoku** (Sudoku) – The sudoku to solve.

Yields *SolveStep* – The next solve step.

class sudokutools.solvers.**NakedTriple**(*clues=()*, *affected=()*, *values=()*)
Bases: *sudokutools.solvers.NakedTuple*

n = 3

class sudokutools.solvers.**NakedTuple**(*clues=()*, *affected=()*, *values=()*)
Bases: *sudokutools.solvers.SolveStep*

Finds naked tuples in a sudoku.

A naked tuple is a set of n fields in a row, column or box, which (in unison) contain a set of at most n candidates.

These candidates can be removed from all fields in the same row, column or box.

build_actions(*sudoku*)

classmethod find(*sudoku*)
Iterates through all possible solve steps of this class.

Parameters **sudoku** (Sudoku) – The sudoku to solve.

Yields *SolveStep* – The next solve step.

n = 2

class sudokutools.solvers.**PointingPair**(*clues=()*, *affected=()*, *values=()*)
Bases: *sudokutools.solvers.PointingTuple*

n = 2

class sudokutools.solvers.**PointingTriple**(*clues=()*, *affected=()*, *values=()*)
Bases: *sudokutools.solvers.PointingTuple*

n = 3

class sudokutools.solvers.**PointingTuple**(*clues=()*, *affected=()*, *values=()*)
Bases: *sudokutools.solvers.SolveStep*

build_actions(*sudoku*)

classmethod find(*sudoku*)
Iterates through all possible solve steps of this class.

Parameters **sudoku** (Sudoku) – The sudoku to solve.

Yields *SolveStep* – The next solve step.

n = 2

class sudokutools.solvers.**SolveStep**(*clues=()*, *affected=()*, *values=()*)
Bases: object

__ge__(*other*)
x.__ge__(y) <==> x>=y

__gt__(*other*)
x.__gt__(y) <==> x>y

__init__(*clues=()*, *affected=()*, *values=()*)
Create a new solve step.

Parameters

- **clues** (*iterable of (int, int)*) – An iterable of (row, col) pairs which cause this step.

- **affected** (*iterable of (int, int)*) – An iterable of (row, col) pairs which are affected by this step.
- **values** (*iterable of int*) – A list of values to apply to the affected fields.

__le__ (*other*)
x.__le__(y) <==> x<=y

__ne__ (*other*)
x.__ne__(y) <==> x!=y

apply (*sudoku*)
Apply this solve step to the sudoku.

classmethod apply_all (*sudoku*)
Apply all possible steps of this class to the sudoku.

build_actions (*sudoku*)

classmethod find (*sudoku*)
Iterates through all possible solve steps of this class.

Parameters **sudoku** (Sudoku) – The sudoku to solve.

Yields *SolveStep* – The next solve step.

class sudokutools.solvers.**Swordfish** (*clues=()*, *affected=()*, *values=()*)
Bases: *sudokutools.solvers.BasicFish*

n = 3

class sudokutools.solvers.**XWing** (*clues=()*, *affected=()*, *values=()*)
Bases: *sudokutools.solvers.BasicFish*

n = 2

class sudokutools.solvers.**_SingleFieldStep** (*row*, *col*, *value*)
Bases: *sudokutools.solvers.SolveStep*

Represents a solve method, which sets a single field.

__init__ (*row*, *col*, *value*)
Create a new solve step.

Parameters

- **clues** (*iterable of (int, int)*) – An iterable of (row, col) pairs which cause this step.
- **affected** (*iterable of (int, int)*) – An iterable of (row, col) pairs which are affected by this step.
- **values** (*iterable of int*) – A list of values to apply to the affected fields.

build_actions (*sudoku*)

classmethod find (*sudoku*)
Iterates through all possible solve steps of this class.

Parameters **sudoku** (Sudoku) – The sudoku to solve.

Yields *SolveStep* – The next solve step.

sudokutools.solvers.**hints** (*sudoku*)
Yield all available solve steps for the current state of a sudoku.

Parameters **sudoku** (Sudoku) – The sudoku to get hints for.

Yields *SolveStep* – A step available for the given sudoku in the current state.

sudokutools.solvers.**solve**(*sudoku*, *report=<function <lambda>>*)

Solve the sudoku and return the solution.

Parameters

- **sudoku** (Sudoku) – The sudoku to solve.
- **report** (*callable*) – A function taking a single argument (the current step), which can be used as a callback.

Returns The solution of the sudoku.

Return type *Sudoku*

2.1.4 sudokutools.analyze - Rate and check sudokus

Rate and check sudokus.

Functions defined here:

- find_conflicts(): Check sudoku for conflicting fields.
- is_solved(): Check, if a sudoku is solved.
- is_unique(): Check if a sudoku has exactly one solution.
- rate(): Return an integer representation of the difficulty of a sudoku.
- **score(): Return an integer representation of the work required to solve** a sudoku.

sudokutools.analyze.**find_conflicts**(*sudoku*, **coords*)

Yield conflicts in sudoku at coords.

If coords is empty all possible coordinates will be searched.

Parameters

- **sudoku** (Sudoku) – The Sudoku instance to check.
- **coords** (*iterable of (int, int)*) – The coordinates to search within.

Yields *((int, int), (int, int), int)* –

tuple of coordinate pairs and the offending value.

E.g.: ((2, 3), (2, 6), 2) indicates, that there is a conflict for the fields (2, 3) and (2, 6) because both of them contain a 2.

sudokutools.analyze.**is_solved**(*sudoku*)

Check, if the sudoku is solved.

Parameters **sudoku** (Sudoku) – The Sudoku instance to check.

Returns Whether or not the sudoku is solved.

Return type bool

sudokutools.analyze.**is_unique**(*sudoku*)

Check if sudoku has exactly one solution.

Parameters **sudoku** (Sudoku) – The Sudoku instance to check.

Returns Whether or not the sudoku is unique.

Return type bool

sudokutools.analyze.**rate**(*sudoku*)
Rate the difficulty of a sudoku and return 0 <= rating <= 10.

Parameters **sudoku** (Sudoku) – The sudoku to rate.

Returns The rating (a value inclusive between 0 and 10).

Return type (int)

Note: Only completely solved sudokus get a rating of 0.

sudokutools.analyze.**score**(*sudoku*)
Return a score for the given sudoku.

The score depends on the number of empty field as well as which solve methods must be used to solve the sudoku.

Parameters **sudoku** (Sudoku) – The sudoku to score.

Returns

The score (a value between 0 and empty * 10, where empty is the number of empty fields in the sudoku).

Return type (int)

2.1.5 sudokutools.generate - Creating new sudokus

Create new sudokus.

Functions defined here:

- create_solution(): Create a complete sudoku without conflicts.
- generate(): Create a new sudoku.
- generate_from_template(): Create a new sudoku given a template pattern.

Warning: Since the functions in this module work using recursion, generating very large Sudokus will likely To be more precise: Python will raise an RecursionError, if `box_width * box_height >= sys.getrecursionlimit()`.

If you really want to generate Sudokus of this size using sudokutools, you have to increase the recursion limit of Python. See: https://stackoverflow.com/questions/3323001/what-is-the-maximum-recursion-depth-in-python-and-how-to-increase-it

sudokutools.generate.**create_solution**(*box_size=(3, 3)*)
Returns a sudoku, without empty or conflicting fields.

Parameters **box_size** (*int, int*) – box width and box height of the filled sudoku. A standard 9x9 sudoku has box_size=(3, 3).

Returns The completely filled Sudoku instance.

Return type *Sudoku*

sudokutools.generate.**generate**(*min_count=0*, *symmetry=None*, *box_size=(3, 3)*)
Generate a sudoku and return it.

Parameters

- **min_count** (*int*) – Number of fields that must be filled at least. Any number above 81 will raise a ValueError, Any number below 17 makes no sense (but will not cause an error), since unique sudokus must have at least 17 filled fields.
- **symmetry** (*str*) – The kind of symmetry that will be created. Possible values are: None (no symmetry), "rotate-90", "rotate-180", "mirror-x", "mirror-y" and "mirror-xy".
- **box_size** (*int, int*) – box_width and box_height of the filled sudoku. A standard 9x9 sudoku has box_size=(3, 3).

Returns The generated `Sudoku` instance.

Return type *Sudoku*

Raises

- ValueError, if symmetry is not a valid argument.
- ValueError, if min_count is larger then len(sudoku).

`sudokutools.generate.`**generate_from_template**(*template*, *tries=100*)

Create a new sudoku from a given template.

Parameters

- **template** (`Sudoku`) – A sudoku, which describes the pattern to use. Every non-zero value of the template will be a filled field in the created solution.
- **tries** (*int*) – The number of tries until we give up. If tries < 0, the function will run, until a solution is found. Take note, that this may deadlock your program, if a solution is not possible.

Returns The created sudoku.

Return type *Sudoku*

Raises `RuntimeError` – if the sudoku couldn't be created, within the given number of tries.

So symmetry isn't enough for you and you want your sudokus to look like your favorite animal? Then this function is for you! generate_from_template takes the pattern from template and returns a valid sudoku, which matches this pattern (if possible).

Creating sudokus from templates is done in two steps:

1. Create a template (Sudoku) from the template string.
2. Hand over this template to this function.

Example for a template string:

```
111111111
100000001
100000001
100111001
100111001
100111001
100000001
100000001
111111111
```

Will create a sudoku like this:

```
1 2 6 | 9 4 8 | 3 7 5
7     |       |     4
3     |       |     6
------+-------+------
9     | 8 1 2 |     3
5     | 3 9 6 |     1
2     | 4 5 7 |     8
------+-------+------
4     |       |     7
8     |       |     2
6 3 7 | 1 2 5 | 4 8 9
```

2.2 License

```
MIT License

Copyright (c) 2017-2019 Maik Messerschmidt

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all
copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
SOFTWARE.
```

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

Python Module Index

S

Index

Symbols

A

B

C

D

E

F

G

H

I

J

N

P

R

S

V

X