

---

# **sudokutools Documentation**

***Release 0.2.0***

**Maik Messerschmidt**

**May 18, 2018**



---

## Contents

---

<b>1</b>	<b>README</b>	<b>3</b>
<b>2</b>	<b>Contents</b>	<b>5</b>
2.1	The sudokutools library . . . . .	5
2.2	License . . . . .	12
<b>3</b>	<b>Indices and tables</b>	<b>15</b>
	<b>Python Module Index</b>	<b>17</b>



Yet another python sudoku library.



# CHAPTER 1

---

## README

---

You can find a short introduction on the [project site](#).





## 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.

### Package modules:

- `sudokutools.generate`: Create new sudokus.
- `sudokutools.solve`: Low-level solving and checking 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:

- `column_of()`: Returns all coordinates in the column of a given field.
- `row_of()`: Returns all coordinates in the row of a given field.
- `square_of()`: Returns all coordinates in the square of a given field.
- `surrounding_of()`: Returns all surrounding coordinates of a given field.

**class** `sudokutools.sudoku.Sudoku`

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, 9).

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

**Return type** `int`

**Raises** `IndexError` – if the given coordinates are not valid.

`__len__` ()

Return the number of non-empty fields in this sudoku.



**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*() .

**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.

**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`.

`sudokutools.sudoku._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.

`sudokutools.sudoku.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*)

`sudokutools.sudoku.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*)

`sudokutools.sudoku.square_of` (*row, col, include=True*)

Return all coordinates in the square 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 square.

**Return type** list of (*int, int*)

`sudokutools.sudoku.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)

## 2.1.2 `sudokutools.solve` - Solving and checking

Low-level solving and checking of sudokus.

**Functions defined here:**

- `bruteforce()`: Solves a sudoku using brute force.
- `calc_candidates()`: Calculates candidates of a field in a sudoku.
- `init_candidates()`: Sets the candidates for all fields in a sudoku.
- `find_conflicts()`: Check sudoku for conflicting fields.
- `is_unique()`: Check if a sudoku has exactly one solution.

`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.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.solve.init_candidates(sudoku)`

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.

`sudokutools.solve.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

## 2.1.3 sudokutools.generate - Creating new sudokus

Create new sudokus.

**Functions defined here:**

- `create_solution()`: Create a complete sudoku without conflicts.
- `generate()`: Create a new sudoku.

`sudokutools.generate.create_solution()`

Returns a sudoku, without empty or conflicting fields.

**Returns** The completely filled Sudoku instance.

**Return type** *Sudoku*

`sudokutools.generate.generate(min_count=17, symmetry=None)`

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”`.

**Returns** The generated Sudoku instance.

**Return type** *Sudoku*

**Raises**

- `ValueError`, if symmetry is not a valid argument.
- `ValueError`, if min\_count > 81.

`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
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

```
1 MIT License
2
3 Copyright (c) 2017-2018 Maik Messerschmidt
4
5 Permission is hereby granted, free of charge, to any person obtaining a copy
```

(continues on next page)



(continued from previous page)

```

6 of this software and associated documentation files (the "Software"), to deal
7 in the Software without restriction, including without limitation the rights
8 to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
9 copies of the Software, and to permit persons to whom the Software is
10 furnished to do so, subject to the following conditions:
11
12 The above copyright notice and this permission notice shall be included in all
13 copies or substantial portions of the Software.
14
15 THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
16 IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
17 FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
18 AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
19 LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
20 OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
21 SOFTWARE.

```



## CHAPTER 3

---

### Indices and tables

---

- `genindex`
- `modindex`
- `search`



### S

- `sudokutools`, [5](#)
- `sudokutools.generate`, [11](#)
- `sudokutools.solve`, [10](#)
- `sudokutools.sudoku`, [5](#)



## Symbols

`__eq__()` (sudokutools.sudoku.Sudoku method), 6  
`__getitem__()` (sudokutools.sudoku.Sudoku method), 6  
`__len__()` (sudokutools.sudoku.Sudoku method), 6  
`__setitem__()` (sudokutools.sudoku.Sudoku method), 7  
`__str__()` (sudokutools.sudoku.Sudoku method), 7  
`_do_bruteforce()` (in module sudokutools.solve), 10  
`_quad_without_row_and_column_of()` (in module sudokutools.sudoku), 9

## B

`bruteforce()` (in module sudokutools.solve), 10

## C

`calc_candidates()` (in module sudokutools.solve), 10  
`column_of()` (in module sudokutools.sudoku), 9  
`copy()` (sudokutools.sudoku.Sudoku method), 7  
`create_solution()` (in module sudokutools.generate), 11

## D

`decode()` (sudokutools.sudoku.Sudoku class method), 7  
`diff()` (sudokutools.sudoku.Sudoku method), 7

## E

`empty()` (sudokutools.sudoku.Sudoku method), 8  
`encode()` (sudokutools.sudoku.Sudoku method), 8

## F

`find_conflicts()` (in module sudokutools.solve), 10

## G

`generate()` (in module sudokutools.generate), 11  
`generate_from_template()` (in module sudokutools.generate), 11  
`get_candidates()` (sudokutools.sudoku.Sudoku method), 8  
`get_number()` (sudokutools.sudoku.Sudoku method), 8

## I

`init_candidates()` (in module sudokutools.solve), 11

`is_unique()` (in module sudokutools.solve), 11

## R

`remove_candidates()` (sudokutools.sudoku.Sudoku method), 8  
`row_of()` (in module sudokutools.sudoku), 9

## S

`set_candidates()` (sudokutools.sudoku.Sudoku method), 8  
`set_number()` (sudokutools.sudoku.Sudoku method), 9  
`square_of()` (in module sudokutools.sudoku), 9  
`Sudoku` (class in sudokutools.sudoku), 5  
`sudokutools` (module), 5  
`sudokutools.generate` (module), 11  
`sudokutools.solve` (module), 10  
`sudokutools.sudoku` (module), 5  
`surrounding_of()` (in module sudokutools.sudoku), 9