tocolib Documentation

Release 0.3.1

Sebastian Wiesendahl <sebastian@wiesendahl.de>

Jan 14, 2019

Contents

A multipurpose utility library for Python 2 and 3.

The library namespace is `tocoli`. For information on available packages, modules and functionality see *Structure*.

CHAPTER 1

Highlights

1.1 Sorting dictionaries

Sort a 'list' of 'dict' by simply defining the *keys* you like to sort by in order from last to first.

Example:

```
>>> dicts = [{'firstname': 'Bob',   'lastname': 'Abel'},
             {'firstname': 'Alice', 'lastname': 'Bond'},
             {'firstname': 'Carol', 'lastname': 'Bond'},
             {'firstname': 'Bob',   'lastname': 'Bond'},
             {'firstname': 'Carol', 'lastname': 'Abel'},
             {'firstname': 'Alice', 'lastname': 'Abel'}]
```

```
>>> from tocoli.sort import sort_dicts_by_value
>>> sort_dicts_by_value(dicts, ['lastname', 'firstname'])
[{'firstname': 'Alice', 'lastname': 'Abel'},
 {'firstname': 'Bob',   'lastname': 'Abel'},
 {'firstname': 'Carol', 'lastname': 'Abel'},
 {'firstname': 'Alice', 'lastname': 'Bond'},
 {'firstname': 'Bob',   'lastname': 'Bond'},
 {'firstname': 'Carol', 'lastname': 'Bond'}]
```

1.2 A Domain Specific Language for intuitive function calls

The `dsl` package provides a coherent style to access the *tocolib* modules and functions as module or static class functions.

Example:

```
>>> from tocoli.dsl import sort
```

```
>>> sort.dicts.by.value(dicts, ['lastname', 'firstname'])
[{'firstname': 'Alice', 'lastname': 'Abel'},
 {'firstname': 'Bob',   'lastname': 'Abel'},
 {'firstname': 'Carol', 'lastname': 'Abel'},
 {'firstname': 'Alice', 'lastname': 'Bond'},
 {'firstname': 'Bob',   'lastname': 'Bond'},
 {'firstname': 'Carol', 'lastname': 'Bond'}]
```

```
>>> sort.dicts.by.similarity(dicts, 'Karol', ['firstname'])
[{'firstname': 'Carol', 'lastname': 'Bond'},
 {'firstname': 'Carol', 'lastname': 'Abel'},
 {'firstname': 'Alice', 'lastname': 'Bond'},
 {'firstname': 'Alice', 'lastname': 'Abel'},
 {'firstname': 'Bob',   'lastname': 'Abel'},
 {'firstname': 'Bob',   'lastname': 'Bond'}]
```

1.3 Powerful mapping

Use recursive mapping to apply functions to nested data structures.

Example:

```
>>> from tocoli.dsl import map
```

```
>>> def upper(item, parent):
        return item.upper()
```

```
>>> map.recursive(dicts, upper)
[{'firstname': 'BOB', 'lastname': 'ABEL'},
 {'firstname': 'ALICE', 'lastname': 'BOND'},
 {'firstname': 'CAROL', 'lastname': 'BOND'},
 {'firstname': 'BOB', 'lastname': 'BOND'},
 {'firstname': 'CAROL', 'lastname': 'ABEL'},
 {'firstname': 'ALICE', 'lastname': 'ABEL'}]
```

```
>>> map_keys = (map.DEFAULT | map.DICT_KEY) ^ map.DICT_VALUE
>>> map.recursive(dicts, upper, map_keys)
[{'FIRSTNAME': 'Bob', 'LASTNAME': 'Abel'},
 {'FIRSTNAME': 'Alice', 'LASTNAME': 'Bond'},
 {'FIRSTNAME': 'Carol', 'LASTNAME': 'Bond'},
 {'FIRSTNAME': 'Bob', 'LASTNAME': 'Bond'},
 {'FIRSTNAME': 'Carol', 'LASTNAME': 'Abel'},
 {'FIRSTNAME': 'Alice', 'LASTNAME': 'Abel'}]
```

CHAPTER 2

What's New

- Changed minimum requirement: passlib>=1.7.0
- The former `dsl` module is now an own subpackage.
- The *keys* parameter notation for sorting functions changed.
- There are new *flags* paramter options for mapping functions.

For more detailed information on current changes check the CHANGELOG.md

CHAPTER 3

Structure

3.1 Namespace

`tocoli` - root The tocolib wraps the `six` library (Python 2 and 3 compatibility utilities) at the root. Thus all `six` packages and modules are also available under the root namespace.

3.2 Subpackages

`dsl` - a domain specific language for tocolib Python, like it should be. The module contains a domain specific language for common functions like filtering, sorting, mapping and more. All functions have a consistent API and results.

3.3 Modules

`auth` - common authetication helpers Its dangerous out there. This module is all about passwords, hashes, salts, tokens and api keys.

`cmp` - compare utilities For those who like to compare apples with pears. Make different data types comparable.

`enc` - encoding functions Encoding without pain. Provides universal encoding functions.

`filter` - filter functions The good ones go into the pot, the bad ones go into your crop. Advanced functions to filter dictionaries or lists of strings.

`fn` - common lambda functions To Be or not to Be: That is the question! Short value extractor functions and more.

`join` - join/reduce/folding functions Bring together what belongs together.

`map` - mapping functions It's still magic even if you know how it's done. Map data by applying any higher-order function to it.

ratio - ratio functions Comparisons make unhappy, but can be quite useful. Provides ratio functions for varios purposes.

regex - regular expression utilities Find what you are searching for. Generate common regular expressions.

sort - sort functions Chuck Norris is able to sort black pens by color. Sort data by value or keys.

spell - spelling utilities Life doesn't come with spell-check, but tocolib does.

test - testing and benchmarking Tests cant prove the absence of bugs. Thus test as good as you can.

type - type conversion utilities What doesn't fit is made to fit. Universal type transformations.

3.4 Contents

3.4.1 tocoli package

Subpackages

tocoli.dsl package

Submodules

tocoli.dsl.filter module

class tocoli.dsl.filter.**dict**

class by

static key(*d*, *keys*, *all_keys=False*)
Returns this 'dict' when it contains at least one of the specified *keys* else None.

static value(*d*, *keywords*, *keys=None*, *all_keys=False*)
Returns this 'dict' when it contains at least one of the keywords as value in the specified *keys* else None.

class tocoli.dsl.filter.**dicts**

class by

static key(*iterable*, *keys*, *all_keys=False*)
Returns a 'list' of 'dict' whose dictionaries contain at least one of the specified *keys*.

static value(*iterable*, *keywords*, *keys=None*, *all_keys=False*)
Returns a 'list' of 'dict' whose dictionaries contain at least one of the keywords as value in the specified *keys*.

class tocoli.dsl.filter.**strings**

class by

static keywords(*iterable*, *keywords*, *case_sensitive=False*)
Filter strings by exact keywords. Returns a new list.

static similarity(*iterable*, *keywords*, *ratio=0.8*, *weights=(1, 4)*, *case_sensitive=False*)
Filter strings by similar keywords. Returns a new list of strings.

tocoli.dsl.join module

class tocoli.dsl.join.**strings**

class by

static keywords(*list*, *keywords*, *join=' '*)
Join strings by keywords. Returns a new list with joined strings.

tocoli.dsl.map module

class tocoli.dsl.map.**string**

static non_accented(*str*)
Returns a non accented string.

tocoli.dsl.sort module

class tocoli.dsl.sort.**dict**

class by

static key(*d*, *reverse=False*)
Returns a 'list' of tuple(key, value).

static value(*d*, *reverse=False*)
Returns a 'list' of tuple(key, value).

class tocoli.dsl.sort.**dicts**

class by

static similarity(*iterable*, *keyword*, *keys*, *values=<function first_kwarg>*, *default=None*, *sequentially=True*, *reverse=False*, *weights=(1, 1)*, *case_sensitive=False*)
Returns a sorted 'list' of 'dict'.

static value(*iterable*, *keys*, *values=<function first_kwarg>*, *default=None*, *sequentially=True*, *reverse=False*)
Sort a list of dictionaries as you like!

Sort a 'list' of 'dict' by value. For simple sorting define *keys* as 'list' of 'str', where the strings are keynames. The functional *values* parameter behaves like the functional *key* parameter from the built-in *sorted* function.

Examples

Simple sort by a key:

```
>>> dicts = [{'name': 'Eve'},
             {'name': 'Alice', 'email': 'alice@example.com'},
             {'email': 'bob@example.com', 'name': 'Bob'}]
```

```
>>> sort_dicts_by_value(dicts, ['name'])
[{'name': 'Alice', 'email': 'alice@example.com'},
 {'email': 'bob@example.com', 'name': 'Bob'},
 {'name': 'Eve'}]
```

Advanced sort for multiple keys and custom `values` function:

```
>>> dicts = [{'price': 100},
             {'price': 50, 'shipping': 40},
             {'shipping': 5, 'price': 55}]
```

```
>>> def total(price, shipping):
        return price + shipping
```

```
>>> sort_dicts_by_value(dicts,
                        ['price', 'shipping'],
                        values=total,
                        default=0,
                        sequentially=False)
[{'price': 55, 'shipping': 5},
 {'price': 50, 'shipping': 40},
 {'price': 100}]
```

Sort dictionaries with non string keys:

```
>>> dicts = [{1: False},
             {'right': True, 1: False},
             {1: True, 'right': True}]
```

```
>>> def both(left, right):
        return left and right
```

```
>>> sort_dicts_by_value(dicts,
                        [{'key': 1, 'alias': 'left'}, 'right'],
                        values=both,
                        default=False,
                        sequentially=False,
                        reverse=True)
[{1: True, 'right': True},
 {1: False},
 {1: False, 'right': True}]
```

Parameters

- **iterable** (*list(dict)*) – The input dictonaries.
- **keys** (*list(string or dict)*) – Sort by defined keys. If you have non string keys then *keys* should be a 'dict' instead of a 'list' with plain keys, which defines an alternative keyname for the non string key (e.g. `keys={'key': 1: alias:`

'one'} instead of keys=[1]). **Note:** Those dictionaries which do not contain any key of *keys* will be removed from the result.
Options:
- 'key': defines the key
- 'alias': alternative keyname for non string key names
- 'sort': sort order (ASCENDING or DESCENDING)

- **values** (function(**kwargs) -> value) – Returns the value to sort by. Defaults to the first value of given keywords (unordered). The *values* function receives the defined keys as **kwargs. Thus it is possible to process the values easily by their name (e.g. lambda firstname, lastname: firstname + ' ' + lastname). **Note:** Take the *default* and the *sequentially* parameter into account, when you specify a custom *values* function. In sequential mode you have to expect different keywords (e.g. lambda **kwargs: ...).
- **default** (*value*) – Default value. Defaults to None. Defines the default value which is passed to the values function if a specified key from keys is not present or None.
- **sequentially** (*bool*) – Run in sequence. Defaults to True. If True each key in keys gets sorted one after the other else runs in batch mode and passes all keys to the values function. **Note:** The keys get sorted in reverse order. Thus the first key gets sorted as last and vice versa.
- **reverse** (*bool*) – Reverse sorting. Defaults to False. Reverses sotring order for each key of *keys* idependently if the *sequentially* parameter is True else reverses the sorted 'list' as whole.

Returns
A 'list' of 'dict'. Note: Those dictionaries which do not contain any key of *keys* will be removed from the result.
Return type list

class tocoli.dsl.sort.**strings**

class by

static similarity(*iterable*, *keyword*, *reverse=False*, *weights=(1, 1)*, *case_sensitive=False*)
Returns a sorted 'list'.

tocoli.dsl.string module

class tocoli.dsl.string.**strip**

static accents(*str*)
Returns a non accented string.

Module contents

tocoli.py2 package

Submodules

tocoli.py2.enc module

tocoli.py2.enc.**decode**(*input*, *encoding='utf-8'*, *errors='strict'*, *detect='utf-8'*)
Decode any string. Returns a decoded 'unicode' otherwise the input.

tocoli.py2.enc.**encode**(*input*, *output_encoding='utf-8'*, *errors='strict'*, *input_encoding='utf-8'*)
Encode any string. Returns an encoded 'str' otherwise the input.

Module contents

tocoli.py3 package

Submodules

tocoli.py3.enc module

tocoli.py3.enc.**decode**(*input*, *encoding='utf-8'*, *errors='strict'*, *detect='utf-8'*)
Decode any string. Returns a decoded 'str' otherwise the input.

tocoli.py3.enc.**encode**(*input*, *output_encoding='utf-8'*, *errors='strict'*, *input_encoding='utf-8'*)
Encode any string. Returns encoded 'bytes' otherwise the input.

Module contents

Submodules

tocoli.auth module

tocoli.auth.**encrypt_api_key**(*user_id*, *secret_key*)
Encrypts a user id into an api key (token). The generated api key has no expiration time.

tocoli.auth.**encrypt_password**(*password*, *rounds=100001*)
Encrypts a password as hash.

tocoli.auth.**encrypt_token**(*user_id*, *secret_key*, *expiration=1800*)
Encrypts a user id into a timed token. Default expiration time is 30 minutes.

tocoli.auth.**generate_salt**(*length=10*, *chars='ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789abcdefghijklmnopqrstu*
Generates a salt. Defaults to a random salt from [A-Z0-9a-z].

tocoli.auth.**verify_api_key**(*api_key*, *secret_key*)
Verifies a given api key (token). Returns the decoded user id or None.

tocoli.auth.**verify_password**(*hash*, *password*)
Verifies a password hash. Returns True on success else False

tocoli.auth.**verify_token**(*token*, *secret_key*)
Verifies a given token. Returns the decoded user id or None.

tocoli.cmp module

tocoli.cmp.**comparable**(*left*, *right*, *case_sensitive=False*)
Makes values comparable. Returns a tuple of values

tocoli.enc module

tocoli.enc.**decode**(*input*, *encoding='utf-8'*, *errors='strict'*, *detect='utf-8'*)
Decode any string. Returns a unicode decoded string otherwise the input.

Use this function to decode an encoded string. Its recommended to use this function as early as possible in your data flow, when you received data from an external resource (e.g. stdin, file, API, â¦). Once decoded you have access to the encoded information and can operate on it. This func- tion is a universal wrapper for the built-in *codecs.decode()* function.

Examples

Default encoding utf-8:

```
>>> decode('cafÃ©')
u'cafÃ©'     # utf-8 decoded string
```

Non default encoding:

```
>>> decode('café', 'latin-1')
u'cafÃ©'     # utf-8 decoded string
```

Advanced codec support:

```
>>> decode('Y2Fmw6k=\n', 'base64')
u'cafÃ©'     # utf-8 decoded string
```

Parameters

- **input** (*str*) – Any string you like to decode. The string can be a byte string or a unicode decoded string. The function makes sure to treat the input type in the correct manner.
- **encoding** (*Optional[str]*) – Input encoding. Defaults to utf-8. Defines the encoding of the input.
- **errors** (*Optional[str]*) – Error handling schemes. Defaults to 'strict' These string values are predefined:

  'strict' - raise a ValueError error (or a subclass) 'ignore' - ignore the character and continue with the next 'replace' - replace with a suitable replacement character;

  Python will use the official U+FFFD REPLACEMENT CHARACTER for the builtin Unicode codecs on decoding and '?' on encoding.

  'xmlcharrefreplace' - Replace with the appropriate XML character reference (only for encoding).

  'backslashreplace' - Replace with backslashed escape sequences (only for encoding).

- **detect** (*Optional[str]*) – Inner encoding. Defaults to utf-8. This parameter defines the recursive decoding type (e.g. contains a *base64* encoded string inside another encoded string, which might be a utf-8 or something else). If the given inner decoding type is not correct, then the function tries to detect the encoding auto- matically.

Returns unicode (python2) str (python3): Encoded byte string.

tocoli.enc.**encode**(*input*, *encoding='utf-8'*, *errors='strict'*, *input_encoding='utf-8'*)

Encode any string. Returns a byte string otherwise the input.

Use this funtion to encode strings you like to write to any kind of output (e.g. stdout, file, API, â¦). This is useful to exchange information in a standardized way (the encoding). This function is a universal wrapper for the built-in *codecs.encode()* function.

Examples

Default usage:

```
>>> encode('cafÃ©')
b'cafÃ©'      # utf-8 encoded byte string
```

Transform encoding:

```
>>> encode('café', 'utf-8', input_encoding='latin-1')
b'cafÃ©'      # utf-8 encoded byte string
```

Advanced codec support:

```
>>> encode('cafÃ©', 'base64')
b'Y2Fmw6k=\n'       # base64 encoded byte string
                    # inner encoding is always utf-8
```

Parameters

- **input** (*str*) – Any string you like to encode. The string can be a byte string or a unicode decoded string. The function makes sure to treat the input type in the correct manner.
- **encoding** (*Optional[str]*) – Output encoding. Defaults to utf-8. Defines the encoding for the resulting byte string.
- **errors** (*Optional[str]*) – Error handling schemes. Defaults to 'strict' These string values are predefined:

  'strict' - raise a ValueError error (or a subclass) 'ignore' - ignore the character and continue with the next 'replace' - replace with a suitable replacement character;

  Python will use the official U+FFFD REPLACEMENT CHARACTER for the builtin Unicode codecs on decoding and '?' on encoding.

  'xmlcharrefreplace' - Replace with the appropriate XML character reference (only for encoding).

  'backslashreplace' - Replace with backslashed escape sequences (only for encoding).

- **input_encoding** (*Optional[str]*) – Given encoding. Defaults to utf-8. Set this parameter if your input is not encoded as utf-8.

Returns str (python2) bytes (python3): Encoded byte string.

tocoli.filter module

tocoli.filter.**clean**(*str*, *include=''*, *alpha='a-zA-Z'*, *numeric='0-9'*)
Filter unwanted characters. Returns the filtered string.

Examples

```
>>> clean('1.20€')
'120'
```

```
>>> clean('1.20€', '.')
'1.20'
```

```
>>> clean('1.20€', '.€')
'1.20€'
```

```
>>> clean('Hello', alpha='a-z')
'ello'
```

Parameters

- **str** (*str*) – input string
- **include** (*str*) – Regular-expression raw string. Defaults to ''. This parameter defines which additional characters to include to the result.
- **alpha** (*str*) – Regular-expression raw string. Defaults to 'a-zA-Z'. This parameter defines which alpha characters should be in the result.
- **numeric** (*str*) – Regular-expression raw string. Defaults to '0-9'. This parameter defines which numeric characters should be in the result.

Returns string which only contains accepted characters.

Return type str

tocoli.filter.**filter_bytes**(*s*, *where*, *encoding='utf-8'*)
Returns bytes.

tocoli.filter.**filter_dict_by_key**(*d*, *keys*, *all_keys=False*)
Returns this 'dict' when it contains at least one of the specified *keys* else None.

tocoli.filter.**filter_dict_by_value**(*d*, *keywords*, *keys=None*, *all_keys=False*)
Returns this 'dict' when it contains at least one of the keywords as value in the specified *keys* else None.

tocoli.filter.**filter_dicts_by_keys**(*iterable*, *keys*, *all_keys=False*)
Returns a 'list' of 'dict' whose dictionaries contain at least one of the specified *keys*.

tocoli.filter.**filter_dicts_by_values**(*iterable*, *keywords*, *keys=None*, *all_keys=False*)
Returns a 'list' of 'dict' whose dictionaries contain at least one of the keywords as value in the specified *keys*.

tocoli.filter.**filter_iter**(*i*, *where*)
Returns a list.

tocoli.filter.**filter_list**(*l*, *where*)
Returns a list.

tocoli.filter.**filter_set**(*s*, *where*)
Returns a set.

tocoli.filter.**filter_string**(*s*, *where*)
Returns a string.

tocoli.filter.**filter_strings_by_keywords**(*iterable*, *keywords*, *case_sensitive=False*)
Filter strings by exact keywords. Returns a new list.

tocoli.filter.**filter_strings_by_similarity**(*iterable*, *keywords*, *ratio=0.8*, *weights=(1, 4)*, *case_sensitive=False*)
Filter strings by similar keywords. Returns a new list of strings.

tocoli.filter.**filter_tuple**(*t*, *where*)
Returns a list.

tocoli.fn module

tocoli.fn.**false**(**args*, ***kwargs*)

tocoli.fn.**first**(*ordered*)

tocoli.fn.**first_arg**(**args*)

tocoli.fn.**first_kwarg**(***kwargs*)

tocoli.fn.**identities**(**args*)

tocoli.fn.**identities_args_kwargs**(**args*, ***kwargs*)

tocoli.fn.**identities_kwargs**(***kwargs*)

tocoli.fn.**identity**(*x*)

tocoli.fn.**identity_values**(**args*, ***kwargs*)

tocoli.fn.**key**(*iterable*, *value=None*, *sort=False*, *reverse=False*, *first=False*, *position=0*)
Generic key getter. Returns one or more keys.

tocoli.fn.**last**(*ordered*)

tocoli.fn.**last_arg**(**args*)

tocoli.fn.**negate**(*x*)

tocoli.fn.**no**(**args*, ***kwargs*)

tocoli.fn.**return_values_as_tuple**(***kwargs*)

tocoli.fn.**second**(*ordered*)

tocoli.fn.**second_arg**(**args*)

tocoli.fn.**third**(*ordered*)

tocoli.fn.**third_arg**(**args*)

tocoli.fn.**true**(**args*, ***kwargs*)

tocoli.fn.**value**(*iterable*, *key=None*, *position=1*)
Generic value getter. Returns containing value.

tocoli.fn.**yes**(**args*, ***kwargs*)

tocoli.join module

tocoli.join.**join_strings_by_keywords**(*list*, *keywords*, *join=' '*)
Join strings by keywords. Returns a new list with joined strings.

tocoli.join.**join_values_as_string**(**args*, ***kwargs*)
Concatenates all values as one string. Returns a string.

tocoli.map module

tocoli.map.**map_to_non_accented_characters**(*str*)
Returns a non accented string.

tocoli.map.**recursive_map**(*item*, *function=<function first_arg>*, *flags=6487*, *parent=None*)
Maps any function recursivly to the item.

tocoli.ratio module

class tocoli.ratio.**Nominator**

MAX = 'max'

MIN = 'min'

tocoli.ratio.**count_equal_chars**(*str1*, *str2*)

tocoli.ratio.**equal**(*str1*, *str2*, *nominator='max'*)
A simple ratio function based on the equality.

tocoli.ratio.**median**(*list*)

tocoli.ratio.**meta**(*str1*, *str2*, *ratios*, *weights*)
A meta ratio function. Returns a weighted meta ratio.

The Wiesendahl ratio is a meta ratio which combines a weighted ratio of given ratio functions.

Parameters

- **str1** (*str*) – first string
- **str2** (*str*) – second string
- **ratios** (*list(function(str, str) -> float)*) – ratio functions This parameter is a list of ratio functions.
- **weights** (*list(float)*) – list of weights Each weight gets applied to its corresponding function.

Returns the combined and weighted meta ratio

Return type float

tocoli.ratio.**similarity**(*str1*, *str2*, *weights=(1, 1)*, *case_sensitive=True*)

tocoli.regex module

class tocoli.regex.**Anchors**

BEGINNING = '^'

END = '$'

NOT_WORD_BOUNDARY = '\\B'

WORD_BOUNDARY = '\\b'

class tocoli.regex.**CharClass**

ANY = '.'

DIGIT = '\\d'

NOT_DIGIT = '\\D'

NOT_WHITESPACE = '\\S'

NOT_WORD = '\\W'

WHITESPACE = '\\s'

WORD = '\\w'

class tocoli.regex.**Escaped**

CARRIAGE_RETURN = '\\r'

FORM_FEED = '\\f'

LINE_FEED = '\\n'

NULL = '\\0'

TAB = '\\t'

VERTICAL_TAB = '\\v'

class tocoli.regex.**Flags**

GLOBAL_SEARCH = 'g'

IGNORE_CASE = 'i'

MULTILINE = 'm'

class tocoli.regex.**Generator**(*options*, *dictionary*)

generate(*input*)

class tocoli.regex.**QuantifiedRe**(*expr=None*)

Bases: *tocoli.regex.Re*

lazy()

class tocoli.regex.**Quantifier**

OPTIONAL = '?'

PLUS = '+'

STAR = '*'

class tocoli.regex.**Re**(*expr=None*)

Bases: unicode

add(*expr*, *escape=False*)

add_backreference(*n*)

add_escaped_control_char(*char*)

add_escaped_hexadecimal(*pair*)

add_escaped_octal(*triplet*)

add_escaped_unicode(*quadruple*)

add_negated_set(*chars*)

add_set(*chars*)

escape()

group(*capturing=True*)

lookahead(*lookup*, *positive=True*)

lookbehind(*lookup*, *positive=True*)

negated_set()

optional()

plus()

quantify(*min*, *max=None*)

set(*chars=None*)

star()

class tocoli.regex.**Substitution**

AFTER_MATCH = "$'"

BEFORE_MATCH = '$`'

DOLLAR = '$$'

MATCH = '$&'

capture_group()

tocoli.regex.**alternate**(*left*, *right*)

tocoli.regex.**backreference**(*n*)

tocoli.regex.**escape**(*expr*)

tocoli.regex.**escape_control_char**(*char*)

tocoli.regex.**escape_hexadecimal**(*pair*)

tocoli.regex.**escape_octal**(*triplet*)

tocoli.regex.**escape_unicode**(*quadruple*)

tocoli.regex.**generate**(*str*, *start=False*, *end=False*, *match=None*, *quantifier='{0, }'*, *setLike=False*, *dictionary=None*)

Generates a python string for regular-expressions.

tocoli.regex.**group**(*expr*, *capturing=True*)

tocoli.regex.**group_capturing**(*expr*)

tocoli.regex.**group_non_capturing**(*expr*)

tocoli.regex.**lazy**(*quantified_expr*)

tocoli.regex.**lookahead**(*expr*, *lookup*, *positive=True*)

tocoli.regex.**lookahead_negative**(*expr*, *lookup*)

tocoli.regex.**lookahead_positive**(*expr*, *lookup*)

tocoli.regex.**lookbehind**(*expr*, *loookup*, *positive=True*)

tocoli.regex.**lookbehind_negative**(*expr*, *lookup*)

tocoli.regex.**lookbehind_positive**(*expr*, *lookup*)

tocoli.regex.**negated_set**(*chars*, *expr=None*)

tocoli.regex.**optional**(*expr*)

tocoli.regex.**plus**(*expr*)

tocoli.regex.**quantify**(*expr*, *min*, *max=None*)

tocoli.regex.**quantify_lazy**(*expr*, *min*, *max=None*)

tocoli.regex.**range**(*start*, *end*)

tocoli.regex.**set**(*chars*, *expr=None*)

tocoli.regex.**star**(*expr*)

tocoli.sort module

tocoli.sort.**sort_bytes**(*b*, *key=None*, *reverse=False*, *encoding='utf-8'*)
Returns bytes.

tocoli.sort.**sort_dict_by_key**(*d*, *reverse=False*)
Returns a 'list' of tuple(key, value).

tocoli.sort.**sort_dict_by_value**(*d*, *reverse=False*)
Returns a 'list' of tuple(key, value).

tocoli.sort.**sort_dicts_by_similarity**(*iterable*, *keyword*, *keys*, *values=<function first_kwarg>*, *default=None*, *sequentially=True*, *reverse=False*, *weights=(1, 1)*, *case_sensitive=False*)
Returns a sorted 'list' of 'dict'.

tocoli.sort.**sort_dicts_by_value**(*iterable*, *keys*, *values=<function first_kwarg>*, *default=None*, *sequentially=True*, *reverse=False*)
Sort a list of dictionaries as you like!

Sort a 'list' of 'dict' by value. For simple sorting define *keys* as 'list' of 'str', where the strings are keynames. The functional *values* parameter behaves like the functional *key* parameter from the built-in *sorted* function.

Examples

Simple sort by a key:

```
>>> dicts = [{'name': 'Eve'},
             {'name': 'Alice', 'email': 'alice@example.com'},
             {'email': 'bob@example.com', 'name': 'Bob'}]
```

```
>>> sort_dicts_by_value(dicts, ['name'])
[{'name': 'Alice', 'email': 'alice@example.com'},
 {'email': 'bob@example.com', 'name': 'Bob'},
 {'name': 'Eve'}]
```

Advanced sort for multiple keys and custom `values` function:

```
>>> dicts = [{'price': 100},
             {'price': 50, 'shipping': 40},
             {'shipping': 5, 'price': 55}]
```

```
>>> def total(price, shipping):
        return price + shipping
```

```
>>> sort_dicts_by_value(dicts,
                        ['price', 'shipping'],
                        values=total,
                        default=0,
                        sequentially=False)
[{'price': 55, 'shipping': 5},
 {'price': 50, 'shipping': 40},
 {'price': 100}]
```

Sort dictionaries with non string keys:

```
>>> dicts = [{1: False},
             {'right': True, 1: False},
             {1: True, 'right': True}]
```

```
>>> def both(left, right):
        return left and right
```

```
>>> sort_dicts_by_value(dicts,
                        [{'key': 1, 'alias': 'left'}, 'right'],
                        values=both,
                        default=False,
                        sequentially=False,
                        reverse=True)
[{1: True, 'right': True},
 {1: False},
 {1: False, 'right': True}]
```

Parameters

- **iterable** (*list(dict)*) – The input dictonaries.
- **keys** (*list(string or dict)*) – Sort by defined keys. If you have non string keys then *keys* should be a 'dict' instead of a 'list' with plain keys, which defines an alternative keyname for the non string key (e.g. `keys={'key': 1: alias: 'one'}` instead of `keys=[1]`). **Note:** Those dictionaries which do not contain any key of *keys* will be removed from the result.

  Options:

  - `'key'`: defines the key
  - `'alias'`: alternative keyname for non string key names

  - `'sort'`: sort order (`ASCENDING` or `DESCENDING`)

- **`values`** (function(**kwargs) -> value) – Returns the value to sort by. Defaults to the first value of given keywords (unordered). The *values* function receives the defined keys as `**kwargs`. Thus it is possible to process the values easily by their name (e.g. `lambda firstname, lastname: firstname + ' ' + lastname`). **Note:** Take the *default* and the *sequentially* parameter into account, when you specify a custom *values* function. In sequential mode you have to expect different keywords (e.g. `lambda **kwargs: ...`).
- **`default`** (*`value`*) – Default value. Defaults to `None`. Defines the default value which is passed to the `values` function if a specified key from `keys` is not present or `None`.
- **`sequentially`** (*`bool`*) – Run in sequence. Defaults to True. If `True` each key in `keys` gets sorted one after the other else runs in batch mode and passes all `keys` to the `values` function. **Note:** The keys get sorted in reverse order. Thus the first key gets sorted as last and vice versa.
- **`reverse`** (*`bool`*) – Reverse sorting. Defaults to False. Reverses sotring order for each key of *keys* idependently if the *sequentially* parameter is `True` else reverses the sorted 'list' as whole.

Returns

A 'list' of 'dict'. Note: Those dictionaries which do not contain any key of *keys* will be removed from the result.

Return type list

`tocoli.sort.`**`sort_iter`**(*i*, *key=None*, *reverse=False*)
Returns a list.

`tocoli.sort.`**`sort_list`**(*l*, *key=None*, *reverse=False*)
Returns a list.

`tocoli.sort.`**`sort_set`**(*s*, *key=None*, *reverse=False*)
Returns a list.

`tocoli.sort.`**`sort_string`**(*s*, *key=None*, *reverse=False*)
Returns a string.

`tocoli.sort.`**`sort_strings_by_similarity`**(*iterable*, *keyword*, *reverse=False*, *weights=(1, 1)*, *case_sensitive=False*)
Returns a sorted 'list'.

`tocoli.sort.`**`sort_tuple`**(*t*, *key=None*, *reverse=False*)
Returns a list.

tocoli.spell module

`class` `tocoli.spell.`**`Dictionary`**

`latin = {'strict': {u'\xe1': u'a|\xe1|\xe4', u'\xfc': u'u|\xfa|ue|\xfce|\xfc', u'tt`

`tocoli.spell.`**`lookup`**(*word*, *dictionary*)

tocoli.test module

class tocoli.test.**Bencher**(*rounds=1*, *collect=False*, *stopwatch=True*, *precision='0.8'*)

bench(*function*, **args*, ***kwargs*)

class tocoli.test.**FnPrinter**(*seperator='-'*, *width=None*)

fnprint(*function*, **args*, ***kwargs*)

tocoli.test.**arguments**(*args=None*, *kwargs=None*)

tocoli.test.**seperator**(*str=''*, *seperator='-'*, *width=80*)

tocoli.type module

tocoli.type.**to_bool**(*input*)
Returns 'bool'.

tocoli.type.**to_integer**(*input*)
Returns 'int'.

tocoli.type.**to_string**(*input*, *encoding='utf-8'*)

Module contents

The tocolib is a multipurpose utility library.

class tocoli.**Py**

class Enc

default = 'utf-8'

preferred = 'utf-8'

class Ver

hex = 34016496

major = 2

micro = 12

minor = 7

releaselevel = 'final'

serial = 0

text = '2.7.12 (default, Nov 19 2016, 06:48:10) \n[GCC 5.4.0 20160609]'

static std(*io=<open file '<stdout>', mode 'w'>*, *encoding='utf-8'*)

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Python Module Index

t

Index

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

V

W

Y