str_util Documentation

Release 0.1.0

Jakob Majkilde

Jan 05, 2019

Contents

1 Installation **3**

2 Usage **5**

3 Functions **7**
- 3.1 Conversion 7
- 3.2 Assertion 7
- 3.3 Modify 7
- 3.4 Extract 8
- 3.5 List operations 8

Python Module Index **21**

String functions for Python 3, inspired by similar Lotus Domino @functions

Features

- Powerful functions to work with both strings and list of strings
- Fully documented: https://stringfunctions.readthedocs.io
- 98% coverage
- MIT License, source code: https://github.com/majkilde/stringfunctions

CHAPTER 1

Installation

Install the latest release from PyPI:

```
pip install str_util
```

CHAPTER 2

Usage

All functions are available directly off the `str_util` package. You may choose to import individual functions by name, or import all.

```
from str_util import word, is_string

def foo(value):
    if is_string( value ):
        return word(value,1)
    return "not a string"
```

CHAPTER 3

Functions

3.1 Conversion

str_util.to_string(value)	Convert a value to a string.
str_util.to_list(value)	Convert a value to a list.
str_util.implode(strings[, separator])	Concatenate all member of a list into a single string by a separating delimiter.

3.2 Assertion

str_util.is_string(value)	Tests the value to determine whether it is a string.
str_util.is_list(value)	Tests the value to determine whether it is a list.
str_util.is_empty(value)	Return true is value is empty or only contains whitespace
str_util.is_member(source_list, search_list)	Check if the source_list is a subset of the search_list
str_util.is_equal(value1, value2[, ignore_case])	Compare two values and returns trues if they are equal
str_util.contains(value, substrings[, ...])	Determine if a string contains any of the substrings
str_util.contains_all(value, substrings[, ...])	Determine if a string contains all of the substring substrings
str_util.compare(string1, string2[, ignore_case])	Compares two strings
str_util.like(string, pattern[, ignore_case])	Matches a string with a pattern

3.3 Modify

str_util.trim(value)	Removes leading, trailing, and redundant spaces/whitespace from a text string, or from each element of a text list.
str_util.propercase(value)	Converts the words in a string to propername capitalization: the first letter of each word becomes uppercase, the rest become lowercase.
str_util.lowercase(value)	Converts a string or list of strings to lowercase.
str_util.replace_substring(source, fromlist, ...)	Replaces specific words in a string or list with new words

3.4 Extract

str_util.left(value, find[, ignore_case])	Searches a string from left to right and returns the leftmost characters of the string.
str_util.left_back(value, find[, ignore_case])	As `left()` but counts/searches from the back
str_util.right(value, find[, ignore_case])	Searches a string from left to right and returns the rightmost characters of the string.
str_util.right_back(value, find[, ignore_case])	Searches a string from the back (right to left) and returns the rightmost characters.
str_util.word(value, number[, separator])	Returns a specified word from a text string.

3.5 List operations

str_util.unique(source_list[, ignore_case])	Removes duplicate values from a list of strings by returning only the first occurrence of each member of the list.
str_util.index_of(value, substring[, ...])	Find the first occurrence of the substring and return the position, If not found, return -1 First character in the string(first element in list has position = 0
str_util.replace(source, fromlist, tolist[, ...])	Performs a search-and-replace operation on a list.
str_util.diff(list1, list2[, ignore_case])	Remove elements in list2 from list1
str_util.union(list1, list2)	Adds two list
str_util.intersection(list1, list2[, ...])	Intersection of the two given list's is a list which consists of all the elements which are common to both list1 and list2.
str_util.sort(source_list[, ignore_case, ...])	**param list source_list** The list to sort

3.5.1 Functions

str_util.**compare**(*string1*, *string2*, *ignore_case=False*)

Compares two strings

Parameters

- **string1** (*str*) – first string
- **string2** (*str*) – second string

- **ignore_case** (*bool*) – Optional. Specify true to ignore case (Default False)

Returns

- string1 is less than string2: return -1
- string1 equals string2: return 0
- string1 is greater than string2: return 1

Return type int

Compare two strings. Banana comes after Apple in the alphabeth and therefor *compare()* return 1 (for greater)

```
>>> compare( 'Banana','Apple')
1
```

These two strings are equal when ignore_case is true

```
>>> compare( "Der Fluß", "DER fluss", ignore_case=True)
0
```

str_util.**contains**(*value*, *substrings*, *ignore_case=False*)

Determine if a string contains any of the substrings

Parameters

- **value** – (str or list) The string you want to search in
- **substrings** – (str or list) The string(s) you want to search for in string.
- **ignore_case** (*bool*) – Optional. Specify True to perform a case-insensitive search (default False)

```
>>> contains( "Hello World", "world")
False
```

```
>>> contains( "Hello World", "wORLd", True)
True
```

```
>>> contains( "Red Blue Yellow Green", ['Black', 'Low'], ignore_case=True)
True
```

```
>>> contains( ['ABC', 'DEF'], ['B'])
True
```

A blank string is always contained >>> contains(“Red Blue Yellow Green”, [‘Rubbish’, ‘’]) True

str_util.**contains_all**(*value*, *substrings*, *ignore_case=False*)

Determine if a string contains all of the substring substrings

Parameters

- **value** – (str or list) The string you want to search in
- **substrings** – (str or list) The string(s) you want to search for in string.
- **ignore_case** (*bool*) – Optional. Specify True to perform a case-insensitive search (default False)

```
>>> contains_all( "Hello World", "Wo")
True
```

```
>>> contains_all( "Hello World", "world", True)
True
```

```
>>> contains_all( "Red Blue Yellow Green", ['Black', 'Red'])
False
```

```
>>> contains_all( "Red Blue Yellow Green", ['LUE', 'red'], True)
True
```

```
>>> contains_all( ["Red Blue", "Yellow Green"], ['Blue', 'red'], True)
True
```

str_util.**diff**(*list1*, *list2*, *ignore_case=False*)

Remove elements in list2 from list1

Parameters

- **list1** (*list or str*) – first list
- **list2** (*list or str*) – second list
- **ignore_case** (*bool*) – Optional. Specify true to ignore case (Default False)

Returns copy of list1 without the elements found in list2

Return type list

```
>>> diff( ['A','B','C'], ['A','D', 'c'])
['B', 'C']
```

```
>>> diff( ['A','B','C'], 'B')
['A', 'C']
```

```
>>> diff( ['A','B','C'], ['A','D', 'c'], ignore_case=True)
['B']
```

str_util.**implode**(*strings*, *separator=''*)

Concatenate all member of a list into a single string by a separating delimiter. Similar to `separator.join(strings)` but doesn't treat a single string as a list

Parameters

- **strings** (*list*) – strings to concatenate
- **separator** (*str*) – Optional. The delimiter (default='')

Returns String

```
>>> implode( ['a','b','c'])
'abc'
```

```
>>> implode( ['Hello','World'], ' ')
'Hello World'
```

```
>>> implode( 'Hi', '.' )
'Hi'
```

str_util.**index_of**(*value*, *substring*, *ignore_case=False*, *reverse=False*)

Find the first occurrence of the substring and return the position, If not found, return -1 First character in the string(first element in list has position = 0

Parameters

- **value** (*str, list*) – the source to search in
- **substring** (*str*) – the substring to search for in the source value
- **ignore_case** (*bool*) – Optional. Specify True to perform a case-insensitive search (default False)
- **reverse** (*bool*) – Optional. Specify True to search backwards (default False)

Returns Position of the first occurrence of the substring in the string or list. Returns 0 if not found

Return type str,list

```
>>> index_of( 'Jakob','a')
1
```

```
>>> index_of( 'Jakob','K')
-1
```

```
>>> index_of( 'Jakob','K', ignore_case=True)
2
```

```
>>> index_of( ['Red', 'Green','Blue'], 'green', ignore_case=True)
1
```

```
>>> index_of( "This is key: FIS", "is", reverse=True)
5
```

```
>>> index_of( "This is key: FIS", "is")
2
```

```
>>> index_of( "This is key: FIS", "is", reverse=True, ignore_case=True)
14
```

str_util.**intersection**(*list1*, *list2*, *ignore_case=False*)

Intersection of the two given list's is a list which consists of all the elements which are common to both list1 and list2.

Parameters

- **list1** (*list or str*) – first list
- **list2** (*list or str*) – second list
- **ignore_case** (*bool*) – Optional. Specify true to ignore case (Default False)

Returns list with common elements

Return type list

```
>>> intersection( ['A','B','C'], ['A','D', 'c'])
['A']
```

```
>>> intersection( ['A','B','C'], ['A','D', 'c'], ignore_case=True)
['A', 'C']
```

```
>>> intersection("Der Fluß", "DER fluss", ignore_case=True)
['Der Fluß']
```

str_util.**is_empty**(*value*)

Return true is value is empty or only contains whitespace

```
>>> is_empty( "    " )
True
```

```
>>> is_empty( None )
True
```

```
>>> is_empty(['  '])
True
```

str_util.**is_equal**(*value1*, *value2*, *ignore_case=False*)

Compare two values and returns trues if they are equal

Parameters

- **value1** (*list or str*) – first list
- **value2** (*list or str*) – second list
- **ignore_case** (*bool*) – Optional. Specify true to ignore case (Default False)

Returns true if the two values is equal

Return type bool

Match with ignore case

```
>>> is_equal("Der Fluß", "DER fluss", ignore_case=True )
True
```

List in random order is still euqal

```
>>> is_equal(['a','b','c'], ['c','b','a'])
True
```

Both list must contain all elements

```
>>> is_equal(['b','c'], ['c','b','a'])
False
```

str_util.**is_list**(*value*)

Tests the value to determine whether it is a list.

Parameters **value** (*any*) –

Returns True of the value is a list (an instance of the list class)

```
>>> is_list( 'Hello' )
False
```

```
>>> is_list( ['Hello'] )
True
```

str_util.**is_member**(*source_list*, *search_list*, *ignore_case=False*)

Check if the source_list is a subset of the search_list

Parameters

- **source_list** (*list or str*) –
- **search_list** (*list or str*) –
- **ignore_case** (*bool*) – Optional. Specify true to ignore case (Default False)

Returns True if all members of the source_list can be found in the search_list

```
>>> is_member('Admin', ['Owner', 'Admin', 'Reader'])
True
```

```
>>> is_member( ['Jakob','Maiken'], ['Maiken','Amalie','Jakob','Ida'])
True
```

str_util.**is_string**(*value*)

Tests the value to determine whether it is a string.

Parameters **value** (*any*) –

Returns True of the value is a string (an instance of the str class)

```
>>> is_string( 'Hello' )
True
```

```
>>> is_string( ['Hello'] )
False
```

str_util.**left**(*value*, *find*, *ignore_case=False*)

Searches a string from left to right and returns the leftmost characters of the string.

Parameters

- **value** (*str or list*) – The string where you want to find the leftmost characters.
- **find** (*str or int*) –
  - [str] a substring to search for. Function returns all characters to the left of *find*
  - [int] number of leftmost chars to return.
- **ignore_case** (*bool*) – Optional. Specify true to ignore case (Default False)

Returns the leftmost characters of string

Return type str or list

Return the first two characters

```
>>> left( "Hello World", 2 )
'He'
```

If number if greater then the length of the string, then the whole string is returned

```
>>> left( "Hello", 10 )
'Hello'
```

Use a negative number to count from the back, just like the *left_back()* function

```
>>> left( "Hello World", -3 )
'Hello Wo'
```

If the *find* string is not found, then an empty string is returned

```
>>> left( "Happy Birthday", "XYZ")
''
```

Return everything until the letter 'l'

```
>>> left( "Hello World", "l")
'He'
```

Also works on list's

```
>>> left( ["Jakob","Majkilde"], 2)
['Ja', 'Ma']
```

str_util.**left_back**(*value*, *find*, *ignore_case=False*)

As *left()* but counts/searches from the back

Parameters

- **value** (*str or list*) – The string where you want to find the leftmost characters.
- **find** (*str or int*) –
  - [str] a substring to search for. Left return all character to the left of *find*
  - [int] return the leftmost characters from the string, skipping the *find* leftmost
- **ignore_case** (*bool*) – Optional. Specify true to ignore case (Default False)

Returns the leftmost characters of string

Return type str or list

Skip the last 3 characters

```
>>> left_back( "Hello World", 3 )
'Hello Wo'
```

If *count* is greater than the length of the string, then return an empty string

```
>>> left_back( "Hello", 10 )
''
```

if *count* is negative, then return the whole string

```
>>> left_back( "Hello World", -2 )
'Hello World'
```

return an empty string if the search string is not found

```
>>> left_back( "Happy Birthday", "XYZ")
''
```

Return leftmost characters until the last occurrence of the letter 'l'

```
>>> left_back( "Hello World", "l")
'Hello Wor'
```

str_util.**like**(*string*, *pattern*, *ignore_case=False*)

Matches a string with a pattern

Parameters

- **string** (*str*) – the value to be tested
- **pattern** (*str*) – the pattern. Use ? for any char or * for any sentence. More info: fnmatch
- **ignore_case** (*bool*) – Optional. Specify true to ignore case (Default False)

Returns True if the *pattern* matches the *string*

Return type bool

```
>>> like( 'Jakob', 'jakob')
False
```

```
>>> like( 'Jakob', 'ja?ob', ignore_case=True)
True
```

```
>>> like( ['Petersen','Pedersen','Peter', 'Olsen'],"Pe?er*" )
[True, True, True, False]
```

str_util.**lowercase**(*value*)

Converts a string or list of strings to lowercase. Like the casefold function, but also works on lists.

Parameters **value** (*str or list*) – the string to convert to lowercase

Returns the source string converted to lowercase

Return type str or list

```
>>> lowercase("Der Fluß")
'der fluss'
```

```
>>> lowercase( ['Green','RED','bluE'])
['green', 'red', 'blue']
```

str_util.**propercase**(*value*)

Converts the words in a string to propername capitalization: the first letter of each word becomes uppercase, the rest become lowercase.

Parameters **value** (*str,list*) – The string you want to convert.

```
>>> propercase('hELLO wORLD')
'Hello World'
```

```
>>> propercase(['blue','RED','very grEEn'])
['Blue', 'Red', 'Very Green']
```

str_util.**replace**(*source*, *fromlist*, *tolist*, *ignore_case=False*)

Performs a search-and-replace operation on a list.

Parameters

- **source** (*list or str*) – The list whose values you want to replace
- **fromlist** (*list or str*) – Values to search for
- **tolist** (*list or str*) – Values to replace with
- **ignore_case** (*bool*) – Optional. Specify true to ignore case (Default False)

Returns new list with replaced values

Return type list

Replace Apple with Microsoft

```
>>> replace( ['Lemon','Apple','Orange'], 'Apple','Microsoft')
['Lemon', 'Microsoft', 'Orange']
```

```
>>> replace( ['red', 'yellow', 'green', 'blue'], ['red', 'green', 'blue'], [
↪'purple', 'silver'] )
['purple', 'yellow', 'silver', 'silver']
```

str_util.**replace_substring**(*source*, *fromlist*, *tolist*, *ignore_case=False*)

Replaces specific words in a string or list with new words

Parameters

- **source** (*list or str*) – Source to be updated with new words
- **fromlist** (*list or str*) – Values to search for
- **tolist** (*list or str*) – Values to replace with
- **ignore_case** (*bool*) – Optional. Specify true to ignore case (Default False)

Returns string/list where all value in *fromlist* is replaced with the corresponding values in *tolist*

Return type list or str

```
>>> replace_substring("Like: I like that you like me", "like", "love")
'Like: I love that you love me'
```

```
>>> replace_substring('I want a hIPpo for my birthday', 'hippo', 'giraffe',␣
↪ignore_case=True)
'I want a giraffe for my birthday'
```

```
>>> replace_substring(['Hello World', 'a b c'], ' ', '_')
['Hello_World', 'a_b_c']
```

```
>>> replace_substring('Odd_looking&text!', ['_','&'], ' ')
'Odd looking text!'
```

```
>>> replace_substring('Encode: &', [' ','&'], ['%20','&amp;'])
'Encode:%20&amp;'
```

```
>>> replace_substring( "I like apples", ["like", "apples"], ["hate", "peaches"])
'I hate peaches'
```

str_util.**right**(*value*, *find*, *ignore_case=False*)

Searches a string from left to right and returns the rightmost characters of the string.

Parameters

- **value** (*str or list*) – The string where you want to find the rightmost characters.
- **find** (*str or int*) –
  - [str] a substring to search for. Function returns all characters to the right of *find*
  - [int] skip the first *count* characters and returns the rest.
- **ignore_case** (*bool*) – Optional. Specify true to ignore case (Default False)

Returns the rightmost characters of string

Return type str or list

Skip the first three characters and return the rest

```
>>> right( "Hello World", 3 )
'lo World'
```

If *count* is greater then the length of the string, then return a blank

```
>>> right( "Hello", 10 )
''
```

If *count* is negative the count from the back - just like *right_back()*

```
>>> right( "Hello World", -2 )
'ld'
```

if the search string is not found, a blank is returned >>> right(“Happy Birthday”, “XYZ”) ‘’

Return all characters to the right of the first occurrence of the letter ‘l’

```
>>> right( "Hello World", "l")
'lo World'
```

Also works on list’s

```
>>> right( ["Jakob","Majkilde"], 'j')
['', 'kilde']
```

str_util.**right_back**(*value*, *find*, *ignore_case=False*)

Searches a string from the back (right to left) and returns the rightmost characters.

Parameters

- **value** (*str or list*) – The string where you want to find the rightmost characters.
- **find** (*str or int*) –
  - [str] a substring to search for. Function returns all characters to the right of the last occurrence of *find*
  - [int] return the *count* characters of the string.
- **ignore_case** (*bool*) – Optional. Specify true to ignore case (Default False)

Returns the rightmost characters of string

Return type str or list

Return the last 3 characters of the string

```
>>> right_back( "Hello World", 3 )
'rld'
```

If *count* is greater than the length of the return, then the whole string is returned

```
>>> right_back( "Hello", 10 )
'Hello'
```

if *count* is negative, then return an empty string

```
>>> right_back( "Hello World", -2 )
'Hello World'
```

Return everything to the right of the last occurrence of the letter 'l'

```
>>> right_back( "Hello World", "l")
'd'
```

Also works on list's

```
>>> right_back( ["Jakob","Majkilde"], 2)
['ob', 'de']
```

str_util.**sort**(*source_list*, *ignore_case=False*, *reverse=False*)

Parameters

- **source_list** (*list*) – The list to sort
- **ignore_case** (*bool*) – Optional. Specify true to ignore case (Default False)
- **reverse** (*bool*) – Optional. Specify True to sort the list in descending order (Default False)

Returns The sorted list

Return type list

```
>>> sort( ['Bad','bored','abe','After'])
['After', 'Bad', 'abe', 'bored']
```

```
>>> sort( ['Bad','bored','abe','After'], ignore_case=True)
['abe', 'After', 'Bad', 'bored']
```

str_util.**to_list**(*value*)

Convert a value to a list. Similar to `list(value)`, but also works on existing lists

Parameters **value** (*any*) – the value to convert

Returns the value converted to a string

```
>>> to_list( "Hello")
['Hello']
```

```
>>> to_list(["Hello"])
['Hello']
```

str_util.**to_string**(*value*)

Convert a value to a string. Same as `str(value)`

Parameters **value** (*any*) – the value to convert

Returns the value converted to a string

```
>>> to_string( 5 )
'5'
```

str_util.**trim**(*value*)

Removes leading, trailing, and redundant spaces/whitespace from a text string, or from each element of a text list.

Parameters **value** (*str,list*) – text or text list

Returns The value, with extra spaces and empty elements removed.

Return type str,list

Remove all redundant whitespace from string >>> trim('A B C ') 'A B C'

Trim all entries in list and remove empty entries >>> trim(['Hello ', ' ', ' World']) ['Hello', 'World']

```
>>> trim( [''])
[]
```

str_util.**union**(*list1*, *list2*)

Adds two list

Parameters

- **list1** (*list or str*) – first list
- **list2** (*list or str*) – second list

Returns new list with all elements from both list1 and list2

Return type list

```
>>> union( ['A','B','C'], ['A','D','c'])
['A', 'B', 'C', 'A', 'D', 'c']
```

```
>>> union( 'Hello', 'World')
['Hello', 'World']
```

str_util.**unique**(*source_list*, *ignore_case=False*)

Removes duplicate values from a list of strings by returning only the first occurrence of each member of the list. :param list source_list: Any text list :param bool ignore_case: Optional. Specify true to ignore case (Default False) :return: List with unique members :rtype: list

```
>>> unique( ['A','B','C','B','A'])
['A', 'B', 'C']
```

```
>>> unique( ['red','green','Red','green'])
['red', 'green', 'Red']
```

```
>>> unique( ['red','green','Red','green'], True)
['red', 'green']
```

str_util.**word**(*value*, *number*, *separator=None*)

Returns a specified word from a text string. Words are by default separated by whitespace. First word in a sentence is number 1

Parameters

- **value** (*str or list*) – the sentence to be scanned
- **number** – A position indicating which word you want returned from string. 1 is the first word in the sentence and -1 is the last word
- **separator** – Optional (default is any whitespace)

Returns the selected word

Return type str or list

Get the second word in a sentence

```
>>> word( "Some text here", 2)
'text'
```

Get the fifth word in a senctence with only three words

```
>>> word( "Some text here", 5)
''
```

Return the last word from a sentence, e.g. the lastname of the username

```
>>> word( "Jakob Majkilde", -1)
'Majkilde'
```

Get the second word in a sentence, using a custom separator

```
>>> word( "North, West, East", 2, ", ")
'West'
```

Also works on list's

```
>>> word( ["North, West, East", 'Scandinavia, UK, China'], 2, ", ")
['West', 'UK']
```

Python Module Index

S

str_util, 8

Index

C

compare() (in module str_util), 8
contains() (in module str_util), 9
contains_all() (in module str_util), 9

D

diff() (in module str_util), 10

I

implode() (in module str_util), 10
index_of() (in module str_util), 10
intersection() (in module str_util), 11
is_empty() (in module str_util), 12
is_equal() (in module str_util), 12
is_list() (in module str_util), 12
is_member() (in module str_util), 13
is_string() (in module str_util), 13

L

left() (in module str_util), 13
left_back() (in module str_util), 14
like() (in module str_util), 15
lowercase() (in module str_util), 15

P

propercase() (in module str_util), 15

R

replace() (in module str_util), 15
replace_substring() (in module str_util), 16
right() (in module str_util), 16
right_back() (in module str_util), 17

S

sort() (in module str_util), 18
str_util (module), 8

T

to_list() (in module str_util), 18
to_string() (in module str_util), 18
trim() (in module str_util), 19

U

union() (in module str_util), 19
unique() (in module str_util), 19

W

word() (in module str_util), 19