When the pattern matches more than one string in the Series, all matches are returned:

```
>>> s.str.findall('on')
0    [on]
1    [on]
2      []
dtype: object
```

Regular expressions are supported too. For instance, the search for all the strings ending with the word 'on' is shown next:

```
>>> s.str.findall('on$')
0    [on]
1      []
2      []
dtype: object
```

If the pattern is found more than once in the same string, then a list of multiple strings is returned:

```
>>> s.str.findall('b')
0        []
1        []
2    [b, b]
dtype: object
```

pandas.Series.str.get

Series.str.**get**(*self*, *i*)

Extract element from each component at specified position.

Extract element from lists, tuples, or strings in each element in the Series/Index.

Parameters

i [int] Position of element to extract.

Returns

Series or Index

Examples

```
>>> s = pd.Series(["String",
...                (1, 2, 3),
...                ["a", "b", "c"],
...                123,
...                -456,
...                {1: "Hello", "2": "World"}])
>>> s
0                       String
1                    (1, 2, 3)
2                    [a, b, c]
3                          123
4                         -456
5    {1: 'Hello', '2': 'World'}
dtype: object
```

```
>>> s.str.get(1)
0        t
1        2
2        b
3      NaN
4      NaN
5    Hello
dtype: object
```

```
>>> s.str.get(-1)
0      g
1      3
2      c
3    NaN
4    NaN
5    None
dtype: object
```

pandas.Series.str.index

`Series.str.`**`index`**(*self*, *sub*, *start=0*, *end=None*)

Return lowest indexes in each strings where the substring is fully contained between [start:end]. This is the same as `str.find` except instead of returning -1, it raises a ValueError when the substring is not found. Equivalent to standard `str.index`.

Parameters

sub [str] Substring being searched.

start [int] Left edge index.

end [int] Right edge index.

Returns

Series or Index of object

See also:

rindex Return highest indexes in each strings.

pandas.Series.str.join

`Series.str.`**`join`**(*self*, *sep*)

Join lists contained as elements in the Series/Index with passed delimiter.

If the elements of a Series are lists themselves, join the content of these lists using the delimiter passed to the function. This function is an equivalent to `str.join()`.

Parameters

sep [str] Delimiter to use between list entries.

Returns

Series/Index: object The list entries concatenated by intervening occurrences of the delimiter.

Raises

AttributeError If the supplied Series contains neither strings nor lists.

See also:

`str.join` Standard library version of this method.
`Series.str.split` Split strings around given separator/delimiter.

Notes

If any of the list items is not a string object, the result of the join will be *NaN*.

Examples

Example with a list that contains non-string elements.

```
>>> s = pd.Series([['lion', 'elephant', 'zebra'],
...                [1.1, 2.2, 3.3],
...                ['cat', np.nan, 'dog'],
...                ['cow', 4.5, 'goat'],
...                ['duck', ['swan', 'fish'], 'guppy']])
>>> s
0         [lion, elephant, zebra]
1                 [1.1, 2.2, 3.3]
2                 [cat, nan, dog]
3                [cow, 4.5, goat]
4    [duck, [swan, fish], guppy]
dtype: object
```

Join all lists using a '-'. The lists containing object(s) of types other than str will produce a NaN.

```
>>> s.str.join('-')
0    lion-elephant-zebra
1                    NaN
2                    NaN
3                    NaN
4                    NaN
dtype: object
```

pandas.Series.str.len

`Series.str.len(self)`

Compute the length of each element in the Series/Index. The element may be a sequence (such as a string, tuple or list) or a collection (such as a dictionary).

Returns

Series or Index of int A Series or Index of integer values indicating the length of each element in the Series or Index.

See also:

`str.len` Python built-in function returning the length of an object.
`Series.size` Returns the length of the Series.

Examples

Returns the length (number of characters) in a string. Returns the number of entries for dictionaries, lists or tuples.

```
>>> s = pd.Series(['dog',
...                '',
...                5,
...                {'foo' : 'bar'},
...                [2, 3, 5, 7],
...                ('one', 'two', 'three')])
>>> s
0                  dog
1
2                    5
3       {'foo': 'bar'}
4         [2, 3, 5, 7]
5    (one, two, three)
dtype: object
>>> s.str.len()
0    3.0
1    0.0
2    NaN
3    1.0
4    4.0
5    3.0
dtype: float64
```

pandas.Series.str.ljust

Series.str.**ljust**(*self*, *width*, *fillchar=' '*)

Filling right side of strings in the Series/Index with an additional character. Equivalent to `str.ljust()`.

Parameters

width [int] Minimum width of resulting string; additional characters will be filled with `fillchar`.

fillchar [str] Additional character for filling, default is whitespace.

Returns

filled [Series/Index of objects.]

pandas.Series.str.lower

Series.str.**lower**(*self*)

Convert strings in the Series/Index to lowercase.

Equivalent to `str.lower()`.

Returns

Series or Index of object

See also:

Series.str.lower Converts all characters to lowercase.
Series.str.upper Converts all characters to uppercase.
Series.str.title Converts first character of each word to uppercase and remaining to lowercase.

Series.str.capitalize Converts first character to uppercase and remaining to lowercase.
Series.str.swapcase Converts uppercase to lowercase and lowercase to uppercase.
Series.str.casefold Removes all case distinctions in the string.

Examples

```
>>> s = pd.Series(['lower', 'CAPITALS', 'this is a sentence', 'SwApCaSe'])
>>> s
0                 lower
1              CAPITALS
2    this is a sentence
3              SwApCaSe
dtype: object
```

```
>>> s.str.lower()
0                 lower
1              capitals
2    this is a sentence
3              swapcase
dtype: object
```

```
>>> s.str.upper()
0                 LOWER
1              CAPITALS
2    THIS IS A SENTENCE
3              SWAPCASE
dtype: object
```

```
>>> s.str.title()
0                 Lower
1              Capitals
2    This Is A Sentence
3              Swapcase
dtype: object
```

```
>>> s.str.capitalize()
0                 Lower
1              Capitals
2    This is a sentence
3              Swapcase
dtype: object
```

```
>>> s.str.swapcase()
0                 LOWER
1              capitals
2    THIS IS A SENTENCE
3              sWaPcAsE
dtype: object
```

pandas.Series.str.lstrip

Series.str.**lstrip**(*self*, *to_strip=None*)

Remove leading and trailing characters.

Strip whitespaces (including newlines) or a set of specified characters from each string in the Series/Index from left side. Equivalent to `str.lstrip()`.

Parameters

to_strip [str or None, default None] Specifying the set of characters to be removed. All combinations of this set of characters will be stripped. If None then whitespaces are removed.

Returns

Series or Index of object

See also:

Series.str.strip Remove leading and trailing characters in Series/Index.
Series.str.lstrip Remove leading characters in Series/Index.
Series.str.rstrip Remove trailing characters in Series/Index.

Examples

```
>>> s = pd.Series(['1. Ant.  ', '2. Bee!\n', '3. Cat?\t', np.nan])
>>> s
0    1. Ant.
1    2. Bee!\n
2    3. Cat?\t
3          NaN
dtype: object
```

```
>>> s.str.strip()
0    1. Ant.
1    2. Bee!
2    3. Cat?
3        NaN
dtype: object
```

```
>>> s.str.lstrip('123.')
0    Ant.
1    Bee!\n
2    Cat?\t
3       NaN
dtype: object
```

```
>>> s.str.rstrip('.!? \n\t')
0    1. Ant
1    2. Bee
2    3. Cat
3       NaN
dtype: object
```

```
>>> s.str.strip('123.!? \n\t')
0    Ant
```

(continues on next page)

(continued from previous page)

```
1    Bee
2    Cat
3    NaN
dtype: object
```

pandas.Series.str.match

Series.str.**match**(*self*, *pat*, *case=True*, *flags=0*, *na=nan*)

Determine if each string matches a regular expression.

Parameters

pat [str] Character sequence or regular expression.

case [bool, default True] If True, case sensitive.

flags [int, default 0 (no flags)] Regex module flags, e.g. re.IGNORECASE.

na [default NaN] Fill value for missing values.

Returns

Series/array of boolean values

See also:

contains Analogous, but less strict, relying on re.search instead of re.match.
extract Extract matched groups.

pandas.Series.str.normalize

Series.str.**normalize**(*self*, *form*)

Return the Unicode normal form for the strings in the Series/Index. For more information on the forms, see the unicodedata.normalize().

Parameters

form [{'NFC', 'NFKC', 'NFD', 'NFKD'}] Unicode form.

Returns

normalized [Series/Index of objects]

pandas.Series.str.pad

Series.str.**pad**(*self*, *width*, *side='left'*, *fillchar=' '*)

Pad strings in the Series/Index up to width.

Parameters

width [int] Minimum width of resulting string; additional characters will be filled with character defined in *fillchar*.

side [{'left', 'right', 'both'}, default 'left'] Side from which to fill resulting string.

fillchar [str, default ' '] Additional character for filling, default is whitespace.

Returns

Series or Index of object Returns Series or Index with minimum number of char in object.

See also:

Series.str.rjust Fills the left side of strings with an arbitrary character. Equivalent to `Series.str.pad(side='left')`.

Series.str.ljust Fills the right side of strings with an arbitrary character. Equivalent to `Series.str.pad(side='right')`.

Series.str.center Fills boths sides of strings with an arbitrary character. Equivalent to `Series.str.pad(side='both')`.

Series.str.zfill Pad strings in the Series/Index by prepending '0' character. Equivalent to `Series.str.pad(side='left', fillchar='0')`.

Examples

```
>>> s = pd.Series(["caribou", "tiger"])
>>> s
0    caribou
1      tiger
dtype: object
```

```
>>> s.str.pad(width=10)
0       caribou
1         tiger
dtype: object
```

```
>>> s.str.pad(width=10, side='right', fillchar='-')
0    caribou---
1    tiger-----
dtype: object
```

```
>>> s.str.pad(width=10, side='both', fillchar='-')
0    -caribou--
1    --tiger---
dtype: object
```

pandas.Series.str.partition

`Series.str.`**`partition`**(*self*, *sep=' '*, *expand=True*)

Split the string at the first occurrence of *sep*.

This method splits the string at the first occurrence of *sep*, and returns 3 elements containing the part before the separator, the separator itself, and the part after the separator. If the separator is not found, return 3 elements containing the string itself, followed by two empty strings.

Parameters

sep [str, default whitespace] String to split on.

expand [bool, default True] If True, return DataFrame/MultiIndex expanding dimensionality. If False, return Series/Index.

Returns

DataFrame/MultiIndex or Series/Index of objects

See also:

rpartition Split the string at the last occurrence of *sep*.

Series.str.split Split strings around given separators.

str.partition Standard library version.

Examples

```
>>> s = pd.Series(['Linda van der Berg', 'George Pitt-Rivers'])
>>> s
0    Linda van der Berg
1    George Pitt-Rivers
dtype: object
```

```
>>> s.str.partition()
        0  1             2
0   Linda     van der Berg
1  George      Pitt-Rivers
```

To partition by the last space instead of the first one:

```
>>> s.str.rpartition()
               0  1            2
0  Linda van der            Berg
1         George     Pitt-Rivers
```

To partition by something different than a space:

```
>>> s.str.partition('-')
                    0  1       2
0  Linda van der Berg
1         George Pitt  -  Rivers
```

To return a Series containing tuples instead of a DataFrame:

```
>>> s.str.partition('-', expand=False)
0    (Linda van der Berg, , )
1    (George Pitt, -, Rivers)
dtype: object
```

Also available on indices:

```
>>> idx = pd.Index(['X 123', 'Y 999'])
>>> idx
Index(['X 123', 'Y 999'], dtype='object')
```

Which will create a MultiIndex:

```
>>> idx.str.partition()
MultiIndex([('X', ' ', '123'),
            ('Y', ' ', '999')],
           dtype='object')
```

Or an index with tuples with `expand=False`:

```
>>> idx.str.partition(expand=False)
Index([('X', ' ', '123'), ('Y', ' ', '999')], dtype='object')
```

pandas.Series.str.repeat

Series.str.**repeat**(*self*, *repeats*)

Duplicate each string in the Series or Index.

Parameters

repeats [int or sequence of int] Same value for all (int) or different value per (sequence).

Returns

Series or Index of object Series or Index of repeated string objects specified by input parameter repeats.

Examples

```
>>> s = pd.Series(['a', 'b', 'c'])
>>> s
0    a
1    b
2    c
dtype: object
```

Single int repeats string in Series

```
>>> s.str.repeat(repeats=2)
0    aa
1    bb
2    cc
dtype: object
```

Sequence of int repeats corresponding string in Series

```
>>> s.str.repeat(repeats=[1, 2, 3])
0      a
1     bb
2    ccc
dtype: object
```

pandas.Series.str.replace

Series.str.**replace**(*self*, *pat*, *repl*, *n=- 1*, *case=None*, *flags=0*, *regex=True*)

Replace occurrences of pattern/regex in the Series/Index with some other string. Equivalent to `str.replace()` or `re.sub()`.

Parameters

pat [str or compiled regex] String can be a character sequence or regular expression.

repl [str or callable] Replacement string or a callable. The callable is passed the regex match object and must return a replacement string to be used. See `re.sub()`.

n [int, default -1 (all)] Number of replacements to make from start.

case [bool, default None] Determines if replace is case sensitive:

- If True, case sensitive (the default if *pat* is a string)
- Set to False for case insensitive

- Cannot be set if *pat* is a compiled regex.

flags [int, default 0 (no flags)] Regex module flags, e.g. re.IGNORECASE. Cannot be set if *pat* is a compiled regex.

regex [bool, default True] Determines if assumes the passed-in pattern is a regular expression:

- If True, assumes the passed-in pattern is a regular expression.
- If False, treats the pattern as a literal string
- Cannot be set to False if *pat* is a compiled regex or *repl* is a callable.

New in version 0.23.0.

Returns

Series or Index of object A copy of the object with all matching occurrences of *pat* replaced by *repl*.

Raises

ValueError

- if *regex* is False and *repl* is a callable or *pat* is a compiled regex
- if *pat* is a compiled regex and *case* or *flags* is set

Notes

When *pat* is a compiled regex, all flags should be included in the compiled regex. Use of *case*, *flags*, or *regex=False* with a compiled regex will raise an error.

Examples

When *pat* is a string and *regex* is True (the default), the given *pat* is compiled as a regex. When *repl* is a string, it replaces matching regex patterns as with `re.sub()`. NaN value(s) in the Series are left as is:

```
>>> pd.Series(['foo', 'fuz', np.nan]).str.replace('f.', 'ba', regex=True)
0    bao
1    baz
2    NaN
dtype: object
```

When *pat* is a string and *regex* is False, every *pat* is replaced with *repl* as with `str.replace()`:

```
>>> pd.Series(['f.o', 'fuz', np.nan]).str.replace('f.', 'ba', regex=False)
0    bao
1    fuz
2    NaN
dtype: object
```

When *repl* is a callable, it is called on every *pat* using `re.sub()`. The callable should expect one positional argument (a regex object) and return a string.

To get the idea:

```
>>> pd.Series(['foo', 'fuz', np.nan]).str.replace('f', repr)
0    <_sre.SRE_Match object; span=(0, 1), match='f'>oo
1    <_sre.SRE_Match object; span=(0, 1), match='f'>uz
2                                                  NaN
dtype: object
```

Reverse every lowercase alphabetic word:

```
>>> repl = lambda m: m.group(0)[::-1]
>>> pd.Series(['foo 123', 'bar baz', np.nan]).str.replace(r'[a-z]+', repl)
0    oof 123
1    rab zab
2        NaN
dtype: object
```

Using regex groups (extract second group and swap case):

```
>>> pat = r"(?P<one>\w+) (?P<two>\w+) (?P<three>\w+)"
>>> repl = lambda m: m.group('two').swapcase()
>>> pd.Series(['One Two Three', 'Foo Bar Baz']).str.replace(pat, repl)
0    tWO
1    bAR
dtype: object
```

Using a compiled regex with flags

```
>>> import re
>>> regex_pat = re.compile(r'FUZ', flags=re.IGNORECASE)
>>> pd.Series(['foo', 'fuz', np.nan]).str.replace(regex_pat, 'bar')
0    foo
1    bar
2    NaN
dtype: object
```

pandas.Series.str.rfind

Series.str.**rfind**(*self*, *sub*, *start=0*, *end=None*)

Return highest indexes in each strings in the Series/Index where the substring is fully contained between [start:end]. Return -1 on failure. Equivalent to standard `str.rfind()`.

Parameters

sub [str] Substring being searched.

start [int] Left edge index.

end [int] Right edge index.

Returns

Series or Index of int.

See also:

find Return lowest indexes in each strings.

pandas.Series.str.rindex

Series.str.**rindex**(*self*, *sub*, *start=0*, *end=None*)

Return highest indexes in each strings where the substring is fully contained between [start:end]. This is the same as `str.rfind` except instead of returning -1, it raises a ValueError when the substring is not found. Equivalent to standard `str.rindex`.

Parameters

sub [str] Substring being searched.

start [int] Left edge index.

end [int] Right edge index.

Returns

Series or Index of object

See also:

index Return lowest indexes in each strings.

pandas.Series.str.rjust

Series.str.**rjust**(*self*, *width*, *fillchar=' '*)

Filling left side of strings in the Series/Index with an additional character. Equivalent to `str.rjust()`.

Parameters

width [int] Minimum width of resulting string; additional characters will be filled with `fillchar`.

fillchar [str] Additional character for filling, default is whitespace.

Returns

filled [Series/Index of objects.]

pandas.Series.str.rpartition

Series.str.**rpartition**(*self*, *sep=' '*, *expand=True*)

Split the string at the last occurrence of *sep*.

This method splits the string at the last occurrence of *sep*, and returns 3 elements containing the part before the separator, the separator itself, and the part after the separator. If the separator is not found, return 3 elements containing two empty strings, followed by the string itself.

Parameters

sep [str, default whitespace] String to split on.

expand [bool, default True] If True, return DataFrame/MultiIndex expanding dimensionality. If False, return Series/Index.

Returns

DataFrame/MultiIndex or Series/Index of objects

See also:

partition Split the string at the first occurrence of *sep*.
Series.str.split Split strings around given separators.
str.partition Standard library version.

Examples

```
>>> s = pd.Series(['Linda van der Berg', 'George Pitt-Rivers'])
>>> s
0    Linda van der Berg
1    George Pitt-Rivers
dtype: object
```

```
>>> s.str.partition()
        0  1             2
0   Linda     van der Berg
1  George      Pitt-Rivers
```

To partition by the last space instead of the first one:

```
>>> s.str.rpartition()
               0  1            2
0  Linda van der            Berg
1         George     Pitt-Rivers
```

To partition by something different than a space:

```
>>> s.str.partition('-')
                    0  1       2
0  Linda van der Berg
1         George Pitt  -  Rivers
```

To return a Series containing tuples instead of a DataFrame:

```
>>> s.str.partition('-', expand=False)
0    (Linda van der Berg, , )
1    (George Pitt, -, Rivers)
dtype: object
```

Also available on indices:

```
>>> idx = pd.Index(['X 123', 'Y 999'])
>>> idx
Index(['X 123', 'Y 999'], dtype='object')
```

Which will create a MultiIndex:

```
>>> idx.str.partition()
MultiIndex([('X', ' ', '123'),
            ('Y', ' ', '999')],
           dtype='object')
```

Or an index with tuples with `expand=False`:

```
>>> idx.str.partition(expand=False)
Index([('X', ' ', '123'), ('Y', ' ', '999')], dtype='object')
```

pandas.Series.str.rstrip

Series.str.**rstrip**(*self*, *to_strip=None*)

Remove leading and trailing characters.

Strip whitespaces (including newlines) or a set of specified characters from each string in the Series/Index from right side. Equivalent to `str.rstrip()`.

Parameters

to_strip [str or None, default None] Specifying the set of characters to be removed. All combinations of this set of characters will be stripped. If None then whitespaces are removed.

Returns

Series or Index of object

See also:

Series.str.strip Remove leading and trailing characters in Series/Index.
Series.str.lstrip Remove leading characters in Series/Index.
Series.str.rstrip Remove trailing characters in Series/Index.

Examples

```
>>> s = pd.Series(['1. Ant.  ', '2. Bee!\n', '3. Cat?\t', np.nan])
>>> s
0    1. Ant.
1    2. Bee!\n
2    3. Cat?\t
3          NaN
dtype: object
```

```
>>> s.str.strip()
0    1. Ant.
1    2. Bee!
2    3. Cat?
3        NaN
dtype: object
```

```
>>> s.str.lstrip('123.')
0    Ant.
1    Bee!\n
2    Cat?\t
3       NaN
dtype: object
```

```
>>> s.str.rstrip('.!? \n\t')
0    1. Ant
1    2. Bee
2    3. Cat
3       NaN
dtype: object
```

```
>>> s.str.strip('123.!? \n\t')
0    Ant
```

(continues on next page)

(continued from previous page)

```
1    Bee
2    Cat
3    NaN
dtype: object
```

pandas.Series.str.slice

Series.str.**slice**(*self*, *start=None*, *stop=None*, *step=None*)

Slice substrings from each element in the Series or Index.

Parameters

start [int, optional] Start position for slice operation.

stop [int, optional] Stop position for slice operation.

step [int, optional] Step size for slice operation.

Returns

Series or Index of object Series or Index from sliced substring from original string object.

See also:

Series.str.slice_replace Replace a slice with a string.

Series.str.get Return element at position. Equivalent to *Series.str.slice(start=i, stop=i+1)* with *i* being the position.

Examples

```
>>> s = pd.Series(["koala", "fox", "chameleon"])
>>> s
0        koala
1          fox
2    chameleon
dtype: object
```

```
>>> s.str.slice(start=1)
0        oala
1          ox
2    hameleon
dtype: object
```

```
>>> s.str.slice(start=-1)
0           a
1           x
2           n
dtype: object
```

```
>>> s.str.slice(stop=2)
0    ko
1    fo
2    ch
dtype: object
```

```
>>> s.str.slice(step=2)
0      kaa
1       fx
2    caeen
dtype: object
```

```
>>> s.str.slice(start=0, stop=5, step=3)
0    kl
1     f
2    cm
dtype: object
```

Equivalent behaviour to:

```
>>> s.str[0:5:3]
0    kl
1     f
2    cm
dtype: object
```

pandas.Series.str.slice_replace

Series.str.**slice_replace**(*self*, *start=None*, *stop=None*, *repl=None*)

Replace a positional slice of a string with another value.

Parameters

start [int, optional] Left index position to use for the slice. If not specified (None), the slice is unbounded on the left, i.e. slice from the start of the string.

stop [int, optional] Right index position to use for the slice. If not specified (None), the slice is unbounded on the right, i.e. slice until the end of the string.

repl [str, optional] String for replacement. If not specified (None), the sliced region is replaced with an empty string.

Returns

Series or Index Same type as the original object.

See also:

Series.str.slice Just slicing without replacement.

Examples

```
>>> s = pd.Series(['a', 'ab', 'abc', 'abdc', 'abcde'])
>>> s
0        a
1       ab
2      abc
3     abdc
4    abcde
dtype: object
```

Specify just *start*, meaning replace *start* until the end of the string with *repl*.

```
>>> s.str.slice_replace(1, repl='X')
0    aX
1    aX
2    aX
3    aX
4    aX
dtype: object
```

Specify just *stop*, meaning the start of the string to *stop* is replaced with *repl*, and the rest of the string is included.

```
>>> s.str.slice_replace(stop=2, repl='X')
0       X
1       X
2      Xc
3     Xdc
4    Xcde
dtype: object
```

Specify *start* and *stop*, meaning the slice from *start* to *stop* is replaced with *repl*. Everything before or after *start* and *stop* is included as is.

```
>>> s.str.slice_replace(start=1, stop=3, repl='X')
0      aX
1      aX
2      aX
3     aXc
4    aXde
dtype: object
```

pandas.Series.str.split

`Series.str.`**`split`**(*self*, *pat=None*, *n=- 1*, *expand=False*)

Split strings around given separator/delimiter.

Splits the string in the Series/Index from the beginning, at the specified delimiter string. Equivalent to `str.split()`.

Parameters

pat [str, optional] String or regular expression to split on. If not specified, split on whitespace.

n [int, default -1 (all)] Limit number of splits in output. `None`, 0 and -1 will be interpreted as return all splits.

expand [bool, default False] Expand the splitted strings into separate columns.

- If `True`, return DataFrame/MultiIndex expanding dimensionality.
- If `False`, return Series/Index, containing lists of strings.

Returns

Series, Index, DataFrame or MultiIndex Type matches caller unless `expand=True` (see Notes).

See also:

Series.str.split Split strings around given separator/delimiter.
Series.str.rsplit Splits string around given separator/delimiter, starting from the right.
Series.str.join Join lists contained as elements in the Series/Index with passed delimiter.

`str.split` Standard library version for split.
`str.rsplit` Standard library version for rsplit.

Notes

The handling of the *n* keyword depends on the number of found splits:

- If found splits > *n*, make first *n* splits only
- If found splits <= *n*, make all splits
- If for a certain row the number of found splits < *n*, append *None* for padding up to *n* if `expand=True`

If using `expand=True`, Series and Index callers return DataFrame and MultiIndex objects, respectively.

Examples

```
>>> s = pd.Series(["this is a regular sentence",
...                "https://docs.python.org/3/tutorial/index.html",
...                np.nan])
0                     this is a regular sentence
1    https://docs.python.org/3/tutorial/index.html
2                                              NaN
dtype: object
```

In the default setting, the string is split by whitespace.

```
>>> s.str.split()
0                   [this, is, a, regular, sentence]
1    [https://docs.python.org/3/tutorial/index.html]
2                                                NaN
dtype: object
```

Without the *n* parameter, the outputs of *rsplit* and *split* are identical.

```
>>> s.str.rsplit()
0                   [this, is, a, regular, sentence]
1    [https://docs.python.org/3/tutorial/index.html]
2                                                NaN
dtype: object
```

The *n* parameter can be used to limit the number of splits on the delimiter. The outputs of *split* and *rsplit* are different.

```
>>> s.str.split(n=2)
0                     [this, is, a regular sentence]
1    [https://docs.python.org/3/tutorial/index.html]
2                                                NaN
dtype: object
```

```
>>> s.str.rsplit(n=2)
0                     [this is a, regular, sentence]
1    [https://docs.python.org/3/tutorial/index.html]
2                                                NaN
dtype: object
```

The *pat* parameter can be used to split by other characters.

```
>>> s.str.split(pat = "/")
0                         [this is a regular sentence]
1    [https:, , docs.python.org, 3, tutorial, index...
2                                                  NaN
dtype: object
```

When using `expand=True`, the split elements will expand out into separate columns. If NaN is present, it is propagated throughout the columns during the split.

```
>>> s.str.split(expand=True)
                                             0     1     2        3
0                                       this    is     a  regular
1  https://docs.python.org/3/tutorial/index.html  None  None     None
2                                          NaN   NaN   NaN      NaN \
             4
0     sentence
1         None
2          NaN
```

For slightly more complex use cases like splitting the html document name from a url, a combination of parameter settings can be used.

```
>>> s.str.rsplit("/", n=1, expand=True)
                                    0           1
0          this is a regular sentence        None
1  https://docs.python.org/3/tutorial  index.html
2                                 NaN         NaN
```

Remember to escape special characters when explicitly using regular expressions.

```
>>> s = pd.Series(["1+1=2"])
```

```
>>> s.str.split(r"\+|=", expand=True)
     0    1    2
0    1    1    2
```

pandas.Series.str.rsplit

`Series.str.`**`rsplit`**(*self*, *pat=None*, *n=- 1*, *expand=False*)

Split strings around given separator/delimiter.

Splits the string in the Series/Index from the end, at the specified delimiter string. Equivalent to `str.rsplit()`.

Parameters

pat [str, optional] String or regular expression to split on. If not specified, split on whitespace.

n [int, default -1 (all)] Limit number of splits in output. `None`, 0 and -1 will be interpreted as return all splits.

expand [bool, default False] Expand the splitted strings into separate columns.

- If `True`, return DataFrame/MultiIndex expanding dimensionality.
- If `False`, return Series/Index, containing lists of strings.

Returns

Series, Index, DataFrame or MultiIndex Type matches caller unless `expand=True` (see Notes).

See also:

Series.str.split Split strings around given separator/delimiter.
Series.str.rsplit Splits string around given separator/delimiter, starting from the right.
Series.str.join Join lists contained as elements in the Series/Index with passed delimiter.
str.split Standard library version for split.
str.rsplit Standard library version for rsplit.

Notes

The handling of the *n* keyword depends on the number of found splits:

- If found splits > *n*, make first *n* splits only
- If found splits <= *n*, make all splits
- If for a certain row the number of found splits < *n*, append *None* for padding up to *n* if `expand=True`

If using `expand=True`, Series and Index callers return DataFrame and MultiIndex objects, respectively.

Examples

```
>>> s = pd.Series(["this is a regular sentence",
...                "https://docs.python.org/3/tutorial/index.html",
...                np.nan])
0                     this is a regular sentence
1    https://docs.python.org/3/tutorial/index.html
2                                              NaN
dtype: object
```

In the default setting, the string is split by whitespace.

```
>>> s.str.split()
0                   [this, is, a, regular, sentence]
1    [https://docs.python.org/3/tutorial/index.html]
2                                                NaN
dtype: object
```

Without the *n* parameter, the outputs of *rsplit* and *split* are identical.

```
>>> s.str.rsplit()
0                   [this, is, a, regular, sentence]
1    [https://docs.python.org/3/tutorial/index.html]
2                                                NaN
dtype: object
```

The *n* parameter can be used to limit the number of splits on the delimiter. The outputs of *split* and *rsplit* are different.

```
>>> s.str.split(n=2)
0                     [this, is, a regular sentence]
1    [https://docs.python.org/3/tutorial/index.html]
2                                                NaN
dtype: object
```

```
>>> s.str.rsplit(n=2)
0                    [this is a, regular, sentence]
1    [https://docs.python.org/3/tutorial/index.html]
2                                                NaN
dtype: object
```

The *pat* parameter can be used to split by other characters.

```
>>> s.str.split(pat = "/")
0                         [this is a regular sentence]
1    [https:, , docs.python.org, 3, tutorial, index...
2                                                  NaN
dtype: object
```

When using `expand=True`, the split elements will expand out into separate columns. If NaN is present, it is propagated throughout the columns during the split.

```
>>> s.str.split(expand=True)
                                           0     1     2        3
0                                       this    is     a  regular
1  https://docs.python.org/3/tutorial/index.html  None  None     None
2                                        NaN   NaN   NaN      NaN \
             4
0     sentence
1         None
2          NaN
```

For slightly more complex use cases like splitting the html document name from a url, a combination of parameter settings can be used.

```
>>> s.str.rsplit("/", n=1, expand=True)
                                    0           1
0          this is a regular sentence        None
1  https://docs.python.org/3/tutorial  index.html
2                                 NaN         NaN
```

Remember to escape special characters when explicitly using regular expressions.

```
>>> s = pd.Series(["1+1=2"])
```

```
>>> s.str.split(r"\+|=", expand=True)
     0    1    2
0    1    1    2
```

pandas.Series.str.startswith

`Series.str.`**`startswith`**(*self*, *pat*, *na=nan*)

Test if the start of each string element matches a pattern.

Equivalent to `str.startswith()`.

Parameters

pat [str] Character sequence. Regular expressions are not accepted.

na [object, default NaN] Object shown if element tested is not a string.

Returns

Series or Index of bool A Series of booleans indicating whether the given pattern matches the start of each string element.

See also:

`str.startswith` Python standard library string method.
`Series.str.endswith` Same as startswith, but tests the end of string.
`Series.str.contains` Tests if string element contains a pattern.

Examples

```
>>> s = pd.Series(['bat', 'Bear', 'cat', np.nan])
>>> s
0     bat
1    Bear
2     cat
3     NaN
dtype: object
```

```
>>> s.str.startswith('b')
0     True
1    False
2    False
3      NaN
dtype: object
```

Specifying *na* to be *False* instead of *NaN*.

```
>>> s.str.startswith('b', na=False)
0     True
1    False
2    False
3    False
dtype: bool
```

pandas.Series.str.strip

`Series.str.`**`strip`**(*self*, *to_strip=None*)

Remove leading and trailing characters.

Strip whitespaces (including newlines) or a set of specified characters from each string in the Series/Index from left and right sides. Equivalent to `str.strip()`.

Parameters

to_strip [str or None, default None] Specifying the set of characters to be removed. All combinations of this set of characters will be stripped. If None then whitespaces are removed.

Returns

Series or Index of object

See also:

`Series.str.strip` Remove leading and trailing characters in Series/Index.
`Series.str.lstrip` Remove leading characters in Series/Index.
`Series.str.rstrip` Remove trailing characters in Series/Index.

Examples

```
>>> s = pd.Series(['1. Ant.  ', '2. Bee!\n', '3. Cat?\t', np.nan])
>>> s
0    1. Ant.
1    2. Bee!\n
2    3. Cat?\t
3          NaN
dtype: object
```

```
>>> s.str.strip()
0    1. Ant.
1    2. Bee!
2    3. Cat?
3        NaN
dtype: object
```

```
>>> s.str.lstrip('123.')
0    Ant.
1    Bee!\n
2    Cat?\t
3       NaN
dtype: object
```

```
>>> s.str.rstrip('.!? \n\t')
0    1. Ant
1    2. Bee
2    3. Cat
3       NaN
dtype: object
```

```
>>> s.str.strip('123.!? \n\t')
0    Ant
1    Bee
2    Cat
3    NaN
dtype: object
```

pandas.Series.str.swapcase

Series.str.**swapcase**(*self*)

Convert strings in the Series/Index to be swapcased.

Equivalent to `str.swapcase()`.

Returns

Series or Index of object

See also:

Series.str.lower	Converts all characters to lowercase.
Series.str.upper	Converts all characters to uppercase.
Series.str.title	Converts first character of each word to uppercase and remaining to lowercase.
Series.str.capitalize	Converts first character to uppercase and remaining to lowercase.
Series.str.swapcase	Converts uppercase to lowercase and lowercase to uppercase.
Series.str.casefold	Removes all case distinctions in the string.

Examples

```
>>> s = pd.Series(['lower', 'CAPITALS', 'this is a sentence', 'SwApCaSe'])
>>> s
0                 lower
1              CAPITALS
2    this is a sentence
3              SwApCaSe
dtype: object
```

```
>>> s.str.lower()
0                 lower
1              capitals
2    this is a sentence
3              swapcase
dtype: object
```

```
>>> s.str.upper()
0                 LOWER
1              CAPITALS
2    THIS IS A SENTENCE
3              SWAPCASE
dtype: object
```

```
>>> s.str.title()
0                 Lower
1              Capitals
2    This Is A Sentence
3              Swapcase
dtype: object
```

```
>>> s.str.capitalize()
0                 Lower
1              Capitals
2    This is a sentence
3              Swapcase
dtype: object
```

```
>>> s.str.swapcase()
0                 LOWER
1              capitals
2    THIS IS A SENTENCE
3              sWaPcAsE
dtype: object
```