3.6.1 String Methods

These are the string methods which both 8-bit strings and Unicode objects support:

capitalize()
Return a copy of the string with only its first character capitalized.

For 8-bit strings, this method is locale-dependent.

center(*width*[, *fillchar*])
Return centered in a string of length *width*. Padding is done using the specified *fillchar* (default is a space). Changed in version 2.4: Support for the *fillchar* argument.

count(*sub*[, *start*[, *end*]])
Return the number of occurrences of substring *sub* in string S[*start*:*end*]. Optional arguments *start* and *end* are interpreted as in slice notation.

decode([*encoding*[, *errors*]])
Decodes the string using the codec registered for *encoding*. *encoding* defaults to the default string encoding. *errors* may be given to set a different error handling scheme. The default is `'strict'`, meaning that encoding errors raise `UnicodeError`. Other possible values are `'ignore'`, `'replace'` and any other name registered via `codecs.register_error`, see section 4.8.1. New in version 2.2. Changed in version 2.3: Support for other error handling schemes added.

encode([*encoding*[,*errors*]])
Return an encoded version of the string. Default encoding is the current default string encoding. *errors* may be given to set a different error handling scheme. The default for *errors* is `'strict'`, meaning that encoding errors raise a `UnicodeError`. Other possible values are `'ignore'`, `'replace'`, `'xmlcharrefreplace'`, `'backslashreplace'` and any other name registered via `codecs.register_error`, see section 4.8.1. For a list of possible encodings, see section 4.8.3. New in version 2.0. Changed in version 2.3: Support for `'xmlcharrefreplace'` and `'backslashreplace'` and other error handling schemes added.

endswith(*suffix*[, *start*[, *end*]])
Return `True` if the string ends with the specified *suffix*, otherwise return `False`. *suffix* can also be a tuple of suffixes to look for. With optional *start*, test beginning at that position. With optional *end*, stop comparing at that position.

Changed in version 2.5: Accept tuples as *suffix*.

expandtabs([*tabsize*])
Return a copy of the string where all tab characters are expanded using spaces. If *tabsize* is not given, a tab size of `8` characters is assumed.

find(*sub*[, *start*[, *end*]])
Return the lowest index in the string where substring *sub* is found, such that *sub* is contained in the range [*start*, *end*]. Optional arguments *start* and *end* are interpreted as in slice notation. Return `-1` if *sub* is not found.

index(*sub*[, *start*[, *end*]])
Like `find()`, but raise `ValueError` when the substring is not found.

isalnum()
Return true if all characters in the string are alphanumeric and there is at least one character, false otherwise.

For 8-bit strings, this method is locale-dependent.

isalpha()
Return true if all characters in the string are alphabetic and there is at least one character, false otherwise.

For 8-bit strings, this method is locale-dependent.

isdigit()
Return true if all characters in the string are digits and there is at least one character, false otherwise.

For 8-bit strings, this method is locale-dependent.

islower()
Return true if all cased characters in the string are lowercase and there is at least one cased character, false otherwise.

For 8-bit strings, this method is locale-dependent.

isspace()
Return true if there are only whitespace characters in the string and there is at least one character, false otherwise.

For 8-bit strings, this method is locale-dependent.

istitle()
Return true if the string is a titlecased string and there is at least one character, for example uppercase characters may only follow uncased characters and lowercase characters only cased ones. Return false otherwise.

For 8-bit strings, this method is locale-dependent.

isupper()
Return true if all cased characters in the string are uppercase and there is at least one cased character, false otherwise.

For 8-bit strings, this method is locale-dependent.

join(*seq*)
Return a string which is the concatenation of the strings in the sequence *seq*. The separator between elements is the string providing this method.

ljust(*width*[, *fillchar*])
Return the string left justified in a string of length *width*. Padding is done using the specified *fillchar* (default is a space). The original string is returned if *width* is less than `len(s)`. Changed in version 2.4: Support for the *fillchar* argument.

lower()
Return a copy of the string converted to lowercase.

For 8-bit strings, this method is locale-dependent.

lstrip([*chars*])
Return a copy of the string with leading characters removed. The *chars* argument is a string specifying the set of characters to be removed. If omitted or `None`, the *chars* argument defaults to removing whitespace. The *chars* argument is not a prefix; rather, all combinations of its values are stripped:

```
>>> '   spacious   '.lstrip()
'spacious   '
>>> 'www.example.com'.lstrip('cmowz.')
'example.com'
```

Changed in version 2.2.2: Support for the *chars* argument.

partition(*sep*)
Split the string at the first occurrence of *sep*, and return a 3-tuple containing the part before the separator, the separator itself, and the part after the separator. If the separator is not found, return a 3-tuple containing the string itself, followed by two empty strings. New in version 2.5.

replace(*old, new*[, *count*])
Return a copy of the string with all occurrences of substring *old* replaced by *new*. If the optional argument *count* is given, only the first *count* occurrences are replaced.

rfind(*sub* [,*start* [,*end*]])

Return the highest index in the string where substring *sub* is found, such that *sub* is contained within s[start,end]. Optional arguments *start* and *end* are interpreted as in slice notation. Return `-1` on failure.

rindex(*sub*[, *start*[, *end*]])
Like `rfind()` but raises `ValueError` when the substring *sub* is not found.

rjust(*width*[, *fillchar*])
Return the string right justified in a string of length *width*. Padding is done using the specified *fillchar* (default is a space). The original string is returned if *width* is less than `len(`*s*`)`. Changed in version 2.4: Support for the *fillchar* argument.

rpartition(*sep*)
Split the string at the last occurrence of *sep*, and return a 3-tuple containing the part before the separator, the separator itself, and the part after the separator. If the separator is not found, return a 3-tuple containing two empty strings, followed by the string itself. New in version 2.5.

rsplit([*sep* [,*maxsplit*]])
Return a list of the words in the string, using *sep* as the delimiter string. If *maxsplit* is given, at most *maxsplit* splits are done, the *rightmost* ones. If *sep* is not specified or `None`, any whitespace string is a separator. Except for splitting from the right, `rsplit()` behaves like `split()` which is described in detail below. New in version 2.4.

rstrip([*chars*])
Return a copy of the string with trailing characters removed. The *chars* argument is a string specifying the set of characters to be removed. If omitted or `None`, the *chars* argument defaults to removing whitespace. The *chars* argument is not a suffix; rather, all combinations of its values are stripped:

```
>>> '   spacious   '.rstrip()
'   spacious'
>>> 'mississippi'.rstrip('ipz')
'mississ'
```

Changed in version 2.2.2: Support for the *chars* argument.

split([*sep* [,*maxsplit*]])
Return a list of the words in the string, using *sep* as the delimiter string. If *maxsplit* is given, at most *maxsplit* splits are done. (thus, the list will have at most *maxsplit*`+1` elements). If *maxsplit* is not specified, then there is no limit on the number of splits (all possible splits are made). Consecutive delimiters are not grouped together and are deemed to delimit empty strings (for example, "`'1,,2'.split(',')`"returns "`['1', '', '2']`"). The *sep* argument may consist of multiple characters (for example, "`'1, 2, 3'.split(', ')`" returns "`['1', '2', '3']`"). Splitting an empty string with a specified separator returns "`['']`".

If *sep* is not specified or is `None`, a different splitting algorithm is applied. First, whitespace characters (spaces, tabs, newlines, returns, and formfeeds) are stripped from both ends. Then, words are separated by arbitrary length strings of whitespace characters. Consecutive whitespace delimiters are treated as a single delimiter ("`'1 2 3'.split()`" returns "`['1', '2', '3']`"). Splitting an empty string or a string consisting of just whitespace returns an empty list.

splitlines([*keepends*])
Return a list of the lines in the string, breaking at line boundaries. Line breaks are not included in the resulting list unless *keepends* is given and true.

startswith(*prefix*[, *start*[, *end*]])
Return `True` if string starts with the *prefix*, otherwise return `False`. *prefix* can also be a tuple of prefixes to look for. With optional *start*, test string beginning at that position. With optional *end*, stop comparing string at that position.

Changed in version 2.5: Accept tuples as *prefix*.

strip([*chars*])
Return a copy of the string with the leading and trailing characters removed. The *chars* argument is a string specifying the set of characters to be removed. If omitted or `None`, the *chars* argument defaults to removing whitespace. The *chars* argument is not a prefix or suffix; rather, all combinations of its values are stripped:

```
>>> '   spacious   '.strip()
'spacious'
>>> 'www.example.com'.strip('cmowz.')
'example'
```

Changed in version 2.2.2: Support for the *chars* argument.

swapcase()
Return a copy of the string with uppercase characters converted to lowercase and vice versa.

For 8-bit strings, this method is locale-dependent.

title()
Return a titlecased version of the string: words start with uppercase characters, all remaining cased characters are lowercase.

For 8-bit strings, this method is locale-dependent.

translate(*table*[, *deletechars*])
Return a copy of the string where all characters occurring in the optional argument *deletechars* are removed, and the remaining characters have been mapped through the given translation table, which must be a string of length 256.

You can use the `maketrans()` helper function in the `string` module to create a translation table.

For Unicode objects, the `translate()` method does not accept the optional *deletechars* argument. Instead, it returns a copy of the *s* where all characters have been mapped through the given translation table which must be a mapping of Unicode ordinals to Unicode ordinals, Unicode strings or `None`. Unmapped characters are left untouched. Characters mapped to `None` are deleted. Note, a more flexible approach is to create a custom character mapping codec using the `codecs` module (see `encodings.cp1251` for an example).

upper()
Return a copy of the string converted to uppercase.

For 8-bit strings, this method is locale-dependent.

zfill(*width*)
Return the numeric string left filled with zeros in a string of length *width*. The original string is returned if *width* is less than `len(`*s*`)`. New in version 2.2.2.
