8. Strings and Things - Python in a Nutshell, 3rd Edition

Methods of String and Bytes Objects

Unicode `str` and `bytes` objects are immutable sequences, as covered in "Strings". All immutable-sequence operations (repetition, concatenation, indexing, and slicing) apply to them, returning an object of the same type. A string or bytes object `s` also supplies several nonmutating methods, as documented in Table 8-1.

Unless otherwise noted, methods are present on objects of either type. In v3, `str` methods return a Unicode string, while methods of `bytes` objects return a bytestring (in v2, type `unicode` stands for a textual string—i.e., Unicode—and type `str` for a bytestring). Terms such as "letters," "whitespace," and so on, refer to the corresponding attributes of the `string` module, covered in "The string Module".

Note

In Table 8-1, for conciseness, we use `sys.maxsize` for integer default values meaning, in practice, "any number, no matter how large."

Table 8-1.

capitalize	`s.capitalize()` Returns a copy of `s` where the first character, if a letter, is uppercase, and all other letters, if any, are lowercase.
casefold	`s.casefold()` `str` only, v3 only. Returns a string processed by the algorithm described in section 3.13 of the Unicode standard. This is similar to *`s`*`.lower` (described later in this list) but also takes into account broader equivalences, such as that between the German lowercase `'ß'` and `'ss'`, and is thus better suited to case-insensitive matching.
center	`s.center(n,fillchar=' ')` Returns a string of length `max(len(s), n)`, with a copy of `s` in the central part, surrounded by equal numbers of copies of character `fillchar` on both sides (e.g., `'ciao'.center(2)` is `'ciao'` and `'x'.center(4,'_')` is `'_x__'`).
count	`s.count(sub,start=0,end=sys.maxsize)` Returns the number of nonoverlapping occurrences of substring `sub` in `s[start:end]`.
decode	`s.decode(encoding='utf-8',errors='strict')` `bytes` only. Returns a `str` object decoded from the bytes *`s`* according to the given encoding. *`errors`* determines how decoding errors are handled. `'strict'` cause errors to raise `UnicodeError` exceptions, `'ignore'` ignores the malformed data, and `'replace'` replaces them with question marks; see "Unicode" for details. Other values can be registered via `codec.register_error()`, covered in Table 8-7.

encode	`s.encode(encoding=None,errors='strict')` `str` only. Returns a `bytes` object obtained from `s` with the given encoding and error handling. See "Unicode" for more details.
endswith	`s.endswith(suffix,start=0,end=sys.maxsize)` Returns `True` when `s[start:end]` ends with string `suffix`; otherwise, `False`. `suffix` can be a tuple of strings, in which case `endswith` returns `True` when `s[start:end]` ends with any one of them.
expandtabs	`s.expandtabs(tabsize=8)` Returns a copy of `s` where each tab character is changed into one or more spaces, with tab stops every `tabsize` characters.
find	`s.find(sub,start=0,end=sys.maxsize)` Returns the lowest index in `s` where substring `sub` is found, such that `sub` is entirely contained in `s[start:end]`. For example, `'banana'.find('na')` is `2`, as is `'banana'.find('na',1)`, while `'banana'.find('na',3)` is `4`, as is `'banana'.find('na',-2)`. `find` returns `-1` when `sub` is not found.
format	`s.format(*args,**kwargs)` `str` only. Formats the positional and named arguments according to formatting instructions contained in the string *s*. See "String Formatting" for further details.
format_map	`s.format_map(mapping)` `str` only, v3 only. Formats the mapping argument according to formatting instructions contained in the string *s*. Equivalent to `s.format(**mapping)` but uses the mapping directly.
index	`s.index(sub,start=0,end=sys.maxsize)` Like `find`, but raises `ValueError` when `sub` is not found.
isalnum	`s.isalnum()` Returns `True` when `len(s)` is greater than `0` and all characters in `s` are letters or digits. When `s` is empty, or when at least one character of `s` is neither a letter nor a digit, `isalnum` returns `False`.
isalpha	`s.isalpha()` Returns `True` when `len(s)` is greater than `0` and all characters in `s` are letters. When `s` is empty, or when at least one character of `s` is not a letter, `isalpha` returns `False`.
isdecimal	`s.isdecimal()` `str` only, v3 only. Returns `True` when `len(s)` is greater than `0` and all characters in *s* can be used to form decimal-radix numbers. This includes Unicode characters defined as Arabic digits.
isdigit	`s.isdigit()` Returns `True` when `len(s)` is greater than `0` and all characters in `s` are digits. When `s` is empty, or when at least one character of `s` is not a digit, `isdigit` returns `False`.

isidentifier	`s.isidentifier()` `str` only, v3 only. Returns `True` when *`s`* is a valid identifier according to the Python language's definition; keywords also satisfy the definition, so, for example, `'class'.isidentifier()` returns `True`.
islower	`s.islower()` Returns `True` when all letters in `s` are lowercase. When `s` contains no letters, or when at least one letter of `s` is uppercase, `islower` returns `False`.
isnumeric	`s.isnumeric()` `str` only, v3 only. Similar to *`s`*`.isdigit()`, but uses a broader definition of numeric characters that includes all characters defined as numeric in the Unicode standard (such as fractions).
isprintable	`s.isprintable()` `str` only, v3 only. Returns `True` when all characters in *`s`* are spaces (`'\x20'`) or are defined in the Unicode standard as printable. Differently from other methods starting with `is`, `''.isprintable()` returns `True`.
isspace	`s.isspace()` Returns `True` when `len(s)` is greater than `0` and all characters in `s` are whitespace. When `s` is empty, or when at least one character of `s` is not whitespace, `isspace` returns `False`.
istitle	`s.istitle()` Returns `True` when letters in `s` are *titlecase*: a capital letter at the start of each contiguous sequence of letters, all other letters lowercase (e.g., `'King Lear'.istitle()` is `True`). When `s` contains no letters, or when at least one letter of `s` violates the titlecase condition, `istitle` returns `False` (e.g., `'1900'.istitle()` and `'Troilus and Cressida'.istitle()` return `False`).
isupper	`s.isupper()` Returns `True` when all letters in `s` are uppercase. When `s` contains no letters, or when at least one letter of `s` is lowercase, `isupper` returns `False`.
join	`s.join(seq)` Returns the string obtained by concatenating the items of `seq`, which must be an iterable whose items are strings, and interposing a copy of `s` between each pair of items (e.g., `''.join(str(x) for x in range(7))` is `'0123456'` and `'x'.join('aeiou')` is `'axexixoxu'`).
ljust	*`s`*`.ljust(`*`n`*`,`*`fillchar`*`=' ')` Returns a string of length `max(len(s),n)`, with a copy of `s` at the start, followed by zero or more trailing copies of character `fillchar`.

lower	`s.lower()` Returns a copy of `s` with all letters, if any, converted to lowercase.
lstrip	`s.lstrip(x=string.whitespace)` Returns a copy of `s`, removing leading characters that are found in string `x`. For example, `'banana'.lstrip('ab')` returns `'nana'`.
replace	`s.replace(old,new,maxsplit=sys.maxsize)` Returns a copy of `s` with the first `maxsplit` (or fewer, if there are fewer) nonoverlapping occurrences of substring `old` replaced by string `new` (e.g., `'banana'.replace('a','e',2)` returns `'benena'`).
rfind	`s.rfind(sub,start=0,end=sys.maxsize)` Returns the highest index in `s` where substring `sub` is found, such that `sub` is entirely contained in `s[start:end]`. `rfind` returns `-1` if `sub` is not found.
rindex	`s.rindex(sub,start=0,end=sys.maxsize)` Like `rfind`, but raises `ValueError` if `sub` is not found.
rjust	`s.rjust(n,fillchar=' ')` Returns a string of length `max(len(s),n)`, with a copy of `s` at the end, preceded by zero or more leading copies of character `fillchar`.
rstrip	`s.rstrip(x=string.whitespace)` Returns a copy of `s`, removing trailing characters that are found in string `x`. For example, `'banana'.rstrip('ab')` returns `'banan'`.

split `s.split(sep=None,maxsplit=sys.maxsize)`

Returns a list `L` of up to `maxsplit+1` strings. Each item of `L` is a "word" from `s`, where string `sep` separates words. When `s` has more than `maxsplit` words, the last item of `L` is the substring of `s` that follows the first `maxsplit` words. When `sep` is `None`, any string of whitespace separates words (e.g., `'four score and seven years'.split(None,3)` is `['four','score','and','seven years']`).

Note the difference between splitting on `None` (any string of whitespace is a separator) and splitting on `' '` (each single space character, *not* other whitespace such as tabs and newlines, and *not* strings of spaces, is a separator). For example:

```
>>> x = 'a  b'      # two spaces between a and b
>>> x.split()
# or, equivalently,
x.split(None)
['a', 'b']
>>> x.split(' ')
['a', '', 'b']
```

In the first case, the two-spaces string in the middle is a single separator; in the second case, each single space is a separator, so that there is an empty string between the two spaces.

splitlines `s.splitlines(keepends=False)`

Like `s.split('\n')`. When `keepends` is true, however, the trailing `'\n'` is included in each item of the resulting list.

startswith `s.startswith(prefix,start=0,end=sys.maxsize)`

Returns `True` when `s[start:end]` starts with string `prefix`; otherwise, `False`. `prefix` can be a tuple of strings, in which case `startswith` returns `True` when `s[start:end]` starts with any one of them.

strip `s.strip(x=string.whitespace)`

Returns a copy of `s`, removing both leading and trailing characters that are found in string `x`. For example, `'banana'.strip('ab')` is `'nan'`.

swapcase `s.swapcase()`

Returns a copy of `s` with all uppercase letters converted to lowercase and vice versa.

title `s.title()`

Returns a copy of `s` transformed to titlecase: a capital letter at the start of each contiguous sequence of letters, with all other letters (if any) lowercase.

translate `s.translate(table)`

Returns a copy of `s` where characters found in `table` are translated or deleted. In v3 (and in v2, when `s` is an instance of `unicode`), `table` is a `dict` whose keys are Unicode ordinals; values are Unicode ordinals, Unicode strings, or `None` (to delete the character)—for example (coded to work both in v2 and v3, with the redundant-in-v3 `u` prefix on strings):

```
print(u'banana'.translate({ord('a'):None,ord('n'):u'ze'}))# prints:
'bzeze'
```

In v2, when `s` is a string of bytes, its `translate` method is quite different—see the online docs. Here are some examples of v2's `translate`:

```
                                                        some
import stringidentity = string.maketrans('','')print('string     '.
translate(identity,'aeiou'))# prints: sm strng
```

The Unicode or v3 equivalent of this would be:

```
                                                      some
no_vowels = dict.fromkeys(ord(x) for x in 'aeiou')print(u'string     '.
translate(no_vowels))# prints: sm strng
```

Here are v2 examples of turning all vowels into *a*'s and also deleting *s*'s:

```
                                                 some
intoas = string.maketrans('eiou','aaaa')print('string     '.translate(
                    sama              some
intoas))# prints: strang     print('string     '.translate(intoas,'s'))
          ama
# prints: trang
```

The Unicode or v3 equivalent of this would be:

```
                                                                 some
intoas = dict.fromkeys((ord(x) for x in 'eiou'), 'a')print(u'string     '
                               sama
.translate(intoas))# prints: strang     intoas_nos = dict(intoas, s='None
          some                                          ama
')print(u'string     '.translate(intoas_nos))# prints: trang
```

upper `s.upper()`

Returns a copy of `s` with all letters, if any, converted to uppercase.