6.4. textwrap — Text wrapping and filling

Source code: Lib/textwrap.py

The textwrap module provides some convenience functions, as well as TextWrapper, the class that does all the work. If you're just wrapping or filling one or two text strings, the convenience functions should be good enough; otherwise, you should use an instance of TextWrapper for efficiency.

textwrap.**wrap**(*text*, *width=70*, ***kwargs*)

Wraps the single paragraph in *text* (a string) so every line is at most *width* characters long. Returns a list of output lines, without final newlines.

Optional keyword arguments correspond to the instance attributes of TextWrapper, documented below. *width* defaults to `70`.

See the TextWrapper.wrap() method for additional details on how wrap() behaves.

textwrap.**fill**(*text*, *width=70*, ***kwargs*)

Wraps the single paragraph in *text*, and returns a single string containing the wrapped paragraph. fill() is shorthand for

```
"\n".join(wrap(text, ...))
```

In particular, fill() accepts exactly the same keyword arguments as wrap().

textwrap.**shorten**(*text*, *width*, ***kwargs*)

Collapse and truncate the given *text* to fit in the given *width*.

First the whitespace in *text* is collapsed (all whitespace is replaced by single spaces). If the result fits in the *width*, it is returned. Otherwise, enough words are dropped from the end so that the remaining words plus the placeholder fit within width:

```
>>> textwrap.shorten("Hello  world!", width=12)
'Hello world!'
>>> textwrap.shorten("Hello  world!", width=11)
'Hello [...]'
>>> textwrap.shorten("Hello world", width=10, placeholder="...
'Hello...'
```

Optional keyword arguments correspond to the instance attributes of TextWrapper, documented below. Note that the whitespace is collapsed before the text is passed to the TextWrapper fill() function, so changing the value of tabsize, expand_tabs, drop_whitespace, and replace_whitespace will have no effect.

New in version 3.4.

textwrap.**dedent**(*text*)

Remove any common leading whitespace from every line in *text*.

This can be used to make triple-quoted strings line up with the left edge of the display, while still presenting them in the source code in indented form.

Note that tabs and spaces are both treated as whitespace, but they are not equal: the lines `"  hello"` and `"\thello"` are considered to have no common leading whitespace.

For example:

```
def test():
    # end first line with \ to avoid the empty line!
    s = '''\
    hello
      world
    '''
    print(repr(s))          # prints '    hello\n      world\n    '
    print(repr(dedent(s)))  # prints 'hello\n  world\n'
```

textwrap.**indent**(*text*, *prefix*, *predicate=None*)

Add *prefix* to the beginning of selected lines in *text*.

Lines are separated by calling `text.splitlines(True)`.

By default, *prefix* is added to all lines that do not consist solely of whitespace (including any line endings).

For example:

```
>>> s = 'hello\n\n \nworld'
>>> indent(s, '  ')
'  hello\n\n \n  world'
```

The optional *predicate* argument can be used to control which lines are indented. For example, it is easy to add *prefix* to even empty and whitespace-only lines:

```
>>> print(indent(s, '+ ', lambda line: True))
+ hello
+
+
+ world
```

New in version 3.3.

wrap(), fill() and shorten() work by creating a TextWrapper instance and calling a single method on it. That instance is not reused, so for applications that process many text strings using wrap() and/or fill(), it may be more efficient to create your own TextWrapper object.

Text is preferably wrapped on whitespaces and right after the hyphens in hyphenated words; only then will long words be broken if necessary, unless TextWrapper.break_long_words is set to false.

class textwrap.**TextWrapper**(***kwargs*)

The TextWrapper constructor accepts a number of optional keyword arguments. Each keyword argument corresponds to an instance attribute, so for example

```
wrapper = TextWrapper(initial_indent="* ")
```

is the same as

```
wrapper = TextWrapper()
wrapper.initial_indent = "* "
```

You can re-use the same TextWrapper object many times, and you can change any of its options through direct assignment to instance attributes between uses.

The TextWrapper instance attributes (and keyword arguments to the constructor) are as follows:

width

(default: 70) The maximum length of wrapped lines. As long as there are no individual words in the input text longer than width, TextWrapper guarantees that no output line will be longer than width characters.

expand_tabs

(default: True) If true, then all tab characters in *text* will be expanded to spaces using the expandtabs() method of *text*.

tabsize

(default: 8) If expand_tabs is true, then all tab characters in *text* will be expanded to zero or more spaces, depending on the current column and the given tab size.

New in version 3.3.

replace_whitespace

(default: True) If true, after tab expansion but before wrapping, the wrap() method will replace each whitespace character with a single space. The whitespace characters replaced are as follows: tab, newline, vertical tab, formfeed, and carriage return (`'\t\n\v\f\r'`).

Note: If `expand_tabs` is false and `replace_whitespace` is true, each tab character will be replaced by a single space, which is *not* the same as tab expansion.

Note: If `replace_whitespace` is false, newlines may appear in the middle of a line and cause strange output. For this reason, text should be split into paragraphs (using `str.splitlines()` or similar) which are wrapped separately.

drop_whitespace

(default: `True`) If true, whitespace at the beginning and ending of every line (after wrapping but before indenting) is dropped. Whitespace at the beginning of the paragraph, however, is not dropped if non-whitespace follows it. If whitespace being dropped takes up an entire line, the whole line is dropped.

initial_indent

(default: `''`) String that will be prepended to the first line of wrapped output. Counts towards the length of the first line. The empty string is not indented.

subsequent_indent

(default: `''`) String that will be prepended to all lines of wrapped output except the first. Counts towards the length of each line except the first.

fix_sentence_endings

(default: `False`) If true, `TextWrapper` attempts to detect sentence endings and ensure that sentences are always separated by exactly two spaces. This is generally desired for text in a monospaced font. However, the sentence detection algorithm is imperfect: it assumes that a sentence ending consists of a lowercase letter followed by one of `'.'`, `'!'`, or

`'?'`, possibly followed by one of `'"'` or `"'"`, followed by a space. One problem with this is algorithm is that it is unable to detect the difference between "Dr." in

```
[...] Dr. Frankenstein's monster [...]
```

and "Spot." in

```
[...] See Spot. See Spot run [...]
```

fix_sentence_endings is false by default.

Since the sentence detection algorithm relies on `string.lowercase` for the definition of "lowercase letter," and a convention of using two spaces after a period to separate sentences on the same line, it is specific to English-language texts.

break_long_words

(default: `True`) If true, then words longer than width will be broken in order to ensure that no lines are longer than width. If it is false, long words will not be broken, and some lines may be longer than width. (Long words will be put on a line by themselves, in order to minimize the amount by which width is exceeded.)

break_on_hyphens

(default: `True`) If true, wrapping will occur preferably on whitespaces and right after hyphens in compound words, as it is customary in English. If false, only whitespaces will be considered as potentially good places for line breaks, but you need to set break_long_words to false if you want truly insecable words. Default behaviour in previous versions was to always allow breaking hyphenated words.

max_lines

(default: `None`) If not `None`, then the output will contain at most *max_lines* lines, with *placeholder* appearing at the end of the output.

New in version 3.4.

placeholder

(default: `' [...]'`) String that will appear at the end of the output text if it has been truncated.

New in version 3.4.

TextWrapper also provides some public methods, analogous to the module-level convenience functions:

wrap(*text*)

Wraps the single paragraph in *text* (a string) so every line is at most width characters long. All wrapping options are taken from instance attributes of the TextWrapper instance. Returns a list of output lines, without final newlines. If the wrapped output has no content, the returned list is empty.

fill(*text*)

Wraps the single paragraph in *text*, and returns a single string containing the wrapped paragraph.