docformatter

Release 1.7.5

Steven Myint

Jul 12, 2023

CONTENTS:

CHAPTER

ONE

HOW TO INSTALL DOCFORMATTER

1.1 Install from PyPI

The latest released version of `docformatter` is available from PyPI. To install it using pip:

```
$ pip install --upgrade docformatter
```

1.1.1 Extras

If you want to use pyproject.toml to configure `docformatter`, you'll need to install with TOML support:

```
$ pip install --upgrade docformatter[tomli]
```

This is only necessary if you are using Python < 3.11. Beginning with Python 3.11, docformatter will utilize `tomllib` from the standard library.

1.2 Install from GitHub

If you'd like to use an unreleased version, you can also use pip to install `docformatter` from GitHub.

```
$ python -m pip install git+https://github.com/PyCQA/docformatter.git@v1.5.0-rc1
```

Replace the tag `v1.5.0-rc1` with a commit SHA to install an untagged version.

CHAPTER

TWO

HOW TO USE DOCFORMATTER

There are several ways you can use `docformatter`. You can use it from the command line, as a file watcher in PyCharm, in your pre-commit checks, and as a GitHub action. However, before you can use `docformatter`, you'll need to install it.

2.1 Use from the Command Line

To use `docformatter` from the command line, simply:

```
$ docformatter name_of_python_file.py
```

`docformatter` recognizes a number of options for controlling how the tool runs as well as how it will treat various patterns in the docstrings. The help output provides a summary of these options:

```
usage: docformatter [-h] [-i | -c] [-d] [-r] [-e [EXCLUDE ...]]
                    [-n [NON-CAP ...]] [-s [style]] [--rest-section-adorns REGEX]
                    [--black] [--wrap-summaries length]
                    [--wrap-descriptions length] [--force-wrap]
                    [--tab-width width] [--blank] [--pre-summary-newline]
                    [--pre-summary-space] [--make-summary-multi-line]
                    [--close-quotes-on-newline] [--range line line]
                    [--docstring-length length length] [--non-strict]
                    [--config CONFIG] [--version] files [files ...]

Formats docstrings to follow PEP 257.

positional arguments:
  files                 files to format or '-' for standard in

optional arguments:
  -h, --help            show this help message and exit
  -i, --in-place        make changes to files instead of printing diffs
  -c, --check           only check and report incorrectly formatted files
  -r, --recursive       drill down directories recursively
  -e, --exclude         in recursive mode, exclude directories and files by names
  -n, --non-cap         list of words not to capitalize when they appear as the
                        first word in the summary

  -s style, --style style
                        the docstring style to use when formatting parameter
```

(continues on next page)

(continued from previous page)

```
                      lists.  One of epytext, sphinx. (default: sphinx)
--rest-section-adorns REGEX
                      regular expression for identifying reST section adornments
                      (default: [!\"#$%&'()*+,-./\\:;<=>?@[]^_`{|}~]{4,})
--black               make formatting compatible with standard black options
                      (default: False)
--wrap-summaries length
                      wrap long summary lines at this length; set to 0 to
                      disable wrapping (default: 79, 88 with --black option)
--wrap-descriptions length
                      wrap descriptions at this length; set to 0 to disable
                      wrapping (default: 72, 88 with --black option)
--force-wrap
                      force descriptions to be wrapped even if it may result
                      in a mess (default: False)
--tab_width width
                      tabs in indentation are this many characters when
                      wrapping lines (default: 1)
--blank
                      add blank line after elaborate description
                      (default: False)
--pre-summary-newline
                      add a newline before one-line or the summary of a
                      multi-line docstring
                      (default: False)
--pre-summary-space
                      add a space between the opening triple quotes and
                      the first word in a one-line or summary line of a
                      multi-line docstring
                      (default: False)
--make-summary-multi-line
                      add a newline before and after a one-line docstring
                      (default: False)
--close-quotes-on-newline
                      place closing triple quotes on a new-line when a
                      one-line docstring wraps to two or more lines
                      (default: False)
--range start_line end_line
                      apply docformatter to docstrings between these lines;
                      line numbers are indexed at 1
--docstring-length min_length max_length
                      apply docformatter to docstrings of given length range
--non-strict
                      do not strictly follow reST syntax to identify lists
                      (see issue #67) (default: False)
--config CONFIG
                      path to file containing docformatter options
                      (default: ./pyproject.toml)
--version
                      show program's version number and exit
```

Possible exit codes from `docformatter`:

- **1** - if any error encountered
- **2** - if it was interrupted
- **3** - if any file needs to be formatted (in `--check` or `--in-place` mode)

2.2 Use as a PyCharm File Watcher

`docformatter` can be configured as a PyCharm file watcher to automatically format docstrings on saving python files.

Head over to `Preferences > Tools > File Watchers`, click the + icon and configure `docformatter` as shown below:

2.3 Use with pre-commit

`docformatter` is configured for pre-commit and can be set up as a hook with the following `.pre-commit-config.yaml` configuration:

```
- repo: https://github.com/PyCQA/docformatter
  rev: v1.6.1
  hooks:
    - id: docformatter
      additional_dependencies: [tomli]
      args: [--in-place --config ./pyproject.toml]
```

You will need to install `pre-commit` and run `pre-commit install`.

Whether you use `args: [--check]` or `args: [--in-place]`, the commit will fail if `docformatter` processes a change. The `--in-place` option fails because pre-commit does a diff check and fails if it detects a hook changed a file. The `--check` option fails because `docformatter` returns a non-zero exit code.

The `additional_dependencies: [tomli]` is only required if you are using `pyproject.toml` for `docformatter`'s configuration.

2.4 Use with GitHub Actions

`docformatter` is one of the tools included in the python-lint-plus action.

CHAPTER

THREE

HOW TO CONFIGURE DOCFORMATTER

The command line options for `docformatter` can also be stored in a configuration file. Currently only `pyproject.toml`, `setup.cfg`, and `tox.ini` are supported. The configuration file can be passed with a full path. For example:

```
$ docformatter --config ~/.secret/path/to/pyproject.toml
```

If no configuration file is explicitly passed, `docformatter` will search the current directory for the supported files and use the first one found. The order of precedence is `pyproject.toml`, `setup.cfg`, then `tox.ini`.

In `pyproject.toml`, add a section `[tool.docformatter]` with options listed using the same name as command line argument. For example:

```
[tool.docformatter]
recursive = true
wrap-summaries = 82
blank = true
```

In `setup.cfg` or `tox.ini`, add a `[docformatter]` section.

```
[docformatter]
recursive = true
wrap-summaries = 82
blank = true
```

Command line arguments will take precedence over configuration file settings. For example, if the following is in your `pyproject.toml`

```
[tool.docformatter]
recursive = true
wrap-summaries = 82
wrap-descriptions = 81
blank = true
```

And you invoke docformatter as follows:

```
$ docformatter --config ~/.secret/path/to/pyproject.toml --wrap-summaries 68
```

Summaries will be wrapped at 68, not 82.

3.1 A Note on Options to Control Styles

There are various `docformatter` options that can be used to control the style of the docstring. These options can be passed on the command line or set in a configuration file. Currently, the style options are:

- `--black`
- `-s` or `--style`

When passing the `--black` option, the following arguments are set automatically:

- `--pre-summary-space` is set to True
- `--wrap-descriptions` is set to 88
- `--wrap-summaries` is set to 88

All of these options can be overridden from the command line or in the configuration file. Further, the `--pre-summary-space` option only inserts a space before the summary when the summary begins with a double quote (“). For example:

`"""This summary gets no space."""` becomes `"""This summary gets no space."""`

and

`""""This" summary does get a space."""` becomes `""" "This" summary does get a space."""`

The `--style` argument takes a string which is the name of the field list style you are using. Currently, only `sphinx` and `epytext` are recognized, but `numpy` and `google` are future styles. For the selected style, each line in the field lists will be wrapped at the `--wrap-descriptions` length as well as any portion of the elaborate description preceding the parameter list. Field lists that don’t follow the passed style will cause the entire elaborate description to be ignored and remain unwrapped.

3.2 A Note on reST Header Adornments Regex

`docformatter-1.7.2` added a new option `--rest-section-adorns`. This allows for setting the characters used as overline and underline adornments for reST section headers. Per the ReStructuredText Markup Specification, the following are all valid adornment characters,

```
! " # $ % & ' ( ) * + , - . / : ; < = > ? @ [ \ ] ^ _ ` { | } ~
```

Thus, the default regular expression `[!\"#$%&'()*+,-./:;<=>?@[\]^_`{|}~]{4,}` looks for any of these characters appearing at least four times in a row. Note that the list of valid adornment characters includes the double quote (”) and the greater-than sign (>). Four repetitions was selected because:

- Docstrings open and close with triple double quotes.
- Doctests begin with >>>.
- It would be rare for a section header to consist of fewer than four characters.

The user can override this default list of characters by passing a regex from the command line or setting the `rest-section-adorns` option in the configuration file. It may be usefule to set this regex to only include the subset of characters you actually use in your docstrings. For example, to only recognize the recommended list in the ReStructuredText Markup Specification, the following regular expression would be used:

```
[=-`:.'"~^_*+#]{4,}
```

CHAPTER

FOUR

DOCFORMATTER REQUIREMENTS

The goal of `docformatter` is to be an autoformatting tool for producing PEP 257 compliant docstrings. This document provides a discussion of the requirements from various sources for `docformatter`. Every effor will be made to keep this document up to date, but this is not a formal requirements document and shouldn't be construed as such.

4.1 PEP 257 Requirements

PEP 257 provides conventions for docstrings. Conventions are general agreements or customs of usage rather than strict engineering requirements. This is appropriate for providing guidance to a broad community. In order to provide a tool for automatically formatting or style checking docstrings, however, some objective criteria is needed. Fortunately, the language of PEP 257 lends itself to defining objective criteria, or requirements, for such tools.

The conventions in PEP 257 define the high-level structure of docstrings:

- How the docstring needs to be formatted.
- What information needs to be in a docstring.

PEP 257 explicitly ignores markup syntax in the docstring; these are style choices left to the individual or organization to enforce. This gives us two categories of requirements in PEP 257. Let's call them *convention* requirements and *methodology* requirements to be consistent with PEP 257 terminology.

An autoformatter should produce docstrings with the proper *convention* so tools such as `Docutils` or `pydocstyle` can process them properly. The contents of a docstring are irrelevant to tools like `Docutils` or `pydocstyle`. An autoformatter may be able to produce some content, but much of the content requirements would be difficult at best to satisfy automatically.

Requirements take one of three types, **shall**, **should**, and **may**. Various sources provide definitions of, and synonyms for, these words. But generally:

- **Shall** represents an absolute.
- **Should** represents a goal.
- **May** represents an option.

Thus, an autoformatting tool:

- Must produce output that satisfies all the *convention* **shall** requirements.
- Ought to provide arguments to allow the user to dictate how each *convention* **should** or **may** requirement is interpreted.
- Would be nice to produce as much output that satisfies the *methodology* requirements.
- Would be nice to provide arguments to allow the user to turn on/off each *methodology* requirement the tool supports.

4.2 Docstring Style

There are at least four "flavors" of docstrings in common use today; Epytext, Sphinx, NumPy, and Google. Each of these docstring flavors follow the PEP 257 *convention* requirements. What differs between the three docstring flavors is the reST syntax used in the field list of the multi-line docstring.

For example, here is how each syntax documents function arguments.

Epytext syntax:

```
@type num_dogs: int
@param num_dogs: the number of dogs
```

Sphinx syntax:

```
:param param1: The first parameter, defaults to 1.
:type: int
```

Google syntax:

```
Args:
    param1 (int): The first parameter.
```

NumPy syntax:

```
Parameters
----------
param1 : int
    The first parameter.
```

Syntax is also important to `Docutils`. An autoformatter should be aware of syntactical directives so they can be placed properly in the structure of the docstring. To accommodate the various syntax flavors used in docstrings, a third requirement category is introduced, *style*.

Another consideration in the *style* category is line wrapping. According to PEP 257, splitting a one-line docstring is to allow "Emacs' `fill-paragraph` command" to be used. The `fill-paragraph` command is a line-wrapping command. Additionally, it would be desirable to wrap docstrings for visual continuity with the code.

NumPy makes a stylistic decision to place a blank line after the long description.

Some code formatting tools also format docstrings. For example, black places a space before a one-line or the summary line when that line begins with a double quote ("). It would be desirable to provide the user an option to have docformatter also insert this space for compatibility.

Thus, an autoformatting tool:

- Ought to provide arguments to allow the user to select the *style* or "flavor" of their choice.
- Ought to provide arguments to allow the user to, as seamlessly as possible, produce output of a compatible *style* with other formatting tools in the eco-system.
- Would be nice to to provide short cut arguments that represent aliases for a commonly used group of *style* arguments.

4.3 Program Control

Finally, how the `docformatter` tool is used should have some user-defined options to accommodate various use-cases. These could best be described as *stakeholder* requirements. An autoformatting tool:

- Ought to provide arguments to allow the user to integrate it into their existing workflow.

4.3.1 Exceptions and Interpretations

As anyone who's ever been involved with turning a set of engineering requirements into a real world product knows, they're never crystal clear and they're always revised along the way. Interpreting and taking exception to the requirements for an aerospace vehicle would be frowned upon without involving the people who wrote the requirements. However, the consequences for a PEP 257 autoformatting tool doing this are slightly less dire. We have confidence the GitHub issue system is the appropriate mechanism if there's a misinterpretation or inappropriate exception taken.

The following items are exceptions or interpretations of the PEP 257 requirements:

- One-line and summary lines can end with any punctuation. `docformatter` will recognize any of [. ! ?]. Exception to requirement PEP_257_4.5; consistent with Google style. See also #56 for situations when this is not desired.
- One-line and summary lines will have the first word capitalized. `docformatter` will capitalize the first word for grammatical correctness. Interpretation of requirement PEP_257_4.5. Some proper nouns are explicitly spelled using a lowercase letter (e.g., `docformatter`). A user option is provided for a list of words to maintain lower case.
- PEP 257 discusses placing closing quotes on a new line in the multi-line section. However, it really makes no sense here as there is no way this condition could be met for a multi-line docstring. Given the basis provided in PEP 257, this requirement really applies to wrapped one-liners. Thus, this is assumed to apply to wrapped one-liners and the closing quotes will be placed on a line by themselves in this case. However, an argument will be provided to allow the user to select their desired behavior. Interpretation of requirement PEP_257_5.5.

These give rise to the *derived* requirement category which would also cover any requirements that must be met for a higher level requirement to be met.

The table below summarizes the requirements for `docformatter`. It includes an ID for reference, the description from PEP 257, which category the requirement falls in, the type of requirement, and whether `docformatter` has implemented the requirement.

Requirement ID's that begin with PEP_257 are taken from PEP 257. Those prefaced with docformatter are un-related to PEP 257.

4.4 Test Suite

Each requirement in the table above should have one or more test in the test suite to verify compliance. Ideally the test docstring will reference the requirement(s) it is verifying to provide traceability.

4.5 Current Implementation

docformatter currently provides the following arguments for interacting with *convention* requirements.

```
--pre-summary-newline [boolean, default False]
    Boolean to indicate whether to place the summary line on the line after
    the opening quotes in a multi-line docstring.  See requirement
    PEP_257_5.2.
```

docformatter currently provides these arguments for *style* requirements.

```
-s, --style [string, default sphinx]
    name of the docstring syntax style to use for formatting parameter
    lists.
--rest-section-adorns [REGEX, default [!\"#$%&'()*+,-./\\:;<=>?@[]^_`{|}~]{4,}]
    regular expression for identifying reST section adornments
-n, --non-cap [string, default []]
    list of words not to capitalize when they appear as the first word in the
    summary
--black [boolean, default False]
    Boolean to indicate whether to format docstrings to be compatible
    with black.
--blank [boolean, default False]
    Boolean to indicate whether to add a blank line after the
    elaborate description.
--close-quotes-on-newline [boolean, default False]
    Boolean to indicate whether to place closing triple quotes on new line
    for wrapped one-line docstrings.
--make-summary-multi-line [boolean, default False]
    Boolean to indicate whether to add a newline before and after a
    one-line docstring.  This option results in non-conventional
    docstrings; violates requirements PEP_257_4.1 and PEP_257_4.3.
--non-strict [boolean, default False]
    Boolean to indicate whether to ignore strict compliance with reST list
    syntax (see issue #67).
--pre-summary-space [boolean, default False]
    Boolean to indicate whether to add a space between the opening triple
    quotes and the first word in a one-line or summary line of a
    multi-line docstring.
--tab-width [integer, defaults to 1]
    Sets the number of characters represented by a tab when line
    wrapping, for Richard Hendricks and others who use tabs instead of
    spaces.
--wrap-descriptions length [integer, default 79]
    Wrap long descriptions at this length.
--wrap-summaries length [integer, default 72]
    Wrap long one-line docstrings and summary lines in multi-line
    docstrings at this length.
```

docformatter currently provides these arguments for *stakeholder* requirements.

```
--check
    Only check and report incorrectly formatted files.
```

(continues on next page)

(continued from previous page)

```
--config CONFIG
    Path to the file containing docformatter options.
--docstring-length min_length max_length
    Only format docstrings that are [min_length, max_length] rows long.
--exclude
    Exclude directories and files by names.
--force-wrap
    Force descriptions to be wrapped even if it may result in a mess.
    This should likely be removed after implementing the syntax option.
--in-place
    Make changes to files instead of printing diffs.
--range start end
    Only format docstrings that are between [start, end] rows in the file.
--recursive
    Drill down directories recursively.
```

4.6 Arguments Needed for Future Releases

The following are new arguments that are needed to implement **should** or **may** *convention* requirements:

```
--wrap-one-line [boolean, default False]
    Boolean to indicate whether to wrap one-line docstrings.  Provides
    option for requirement PEP_257_4.1.
```

4.7 Issue and Version Management

As bug reports and feature requests arise in the GitHub issue system, these will need to be prioritized. The requirement categories, coupled with the urgency of the issue reported can be used to provide the general prioritization scheme:

- Priority 1: *convention* **bug**
- Priority 2: *style* **bug**
- Priority 3: *stakeholder* **bug**
- Priority 4: *convention* **enhancement**
- Priority 5: *style* **enhancement**
- Priority 6: *stakeholder* **enhancement**
- Priority 7: **chore**

Integration of a bug fix will result in a patch version bump (i.e., 1.5.0 -> 1.5.1). Integration of one or more enhancements will result in a minor version bump (i.e., 1.5.0 -> 1.6.0). One or more release candidates will be provided for each minor or major version bump. These will be indicated by appending *-rcX* to the version number, where the X is the release candidate number beginning with 1. Release candidates will not be uploaded to PyPi, but will be made available via GitHub Releases.

CHAPTER

FIVE

KNOWN ISSUES AND IDIOSYNCRASIES

There are some know issues or idiosyncrasies when using `docformatter`. These are stylistic issues and are in the process of being addressed.

5.1 Wrapping Descriptions

`docformatter` will wrap descriptions, but only in simple cases. If there is text that seems like a bulleted/numbered list, `docformatter` will leave the description as is:

```
- Item one.
- Item two.
- Item three.
```

This prevents the risk of the wrapping turning things into a mess. To force even these instances to get wrapped use `--force-wrap`. This is being addressed by the constellation of issues related to the various syntaxes used in docstrings.

CHAPTER

SIX

AUTHORS

6.1 Author

Steven Myint <git@stevenmyint.com>

6.2 Additional contributions by (sorted by name)

- Alec Merdler <alecmerdler@gmail.com>
- Alexander Biggs <akbiggs@users.noreply.github.com>
- Alexander Kapshuna <kapsh@kap.sh>
- Andrew Howe <howeaj@users.noreply.github.com>
- Andy Hayden <andyhayden1@gmail.com>
- Anthony Sottile <asottile@umich.edu>
- Antoine Dechaume <AntoineD@users.noreply.github.com>
- Asher Foa <asher@toolchain.com>
- Benjamin Schubert <contact@benschubert.me>
- Doyle Rowland <doyle.rowland@reliaqual.com>
- Eric Hutton <mcflugen@users.noreply.github.com>
- Filip Kucharczyk <filip.m.kucharczyk@gmail.com>
- Kapshuna Alexander <kapsh@kap.sh>
- Kian-Meng Ang <kianmeng.ang@gmail.com>
- KotlinIsland <65446343+KotlinIsland@users.noreply.github.com>
- Lisha Li <65045844+lli-fincad@users.noreply.github.com>
- Manuel Kaufmann <humitos@gmail.com>
- Oliver Sieweke <oliver.sieweke@protonmail.com>
- Paul Angerer <48882462+dabauxi@users.noreply.github.com>
- Paul Angerer <48882462+etimoz@users.noreply.github.com>
- Peter Boothe <pboothe@pboothe2.nyc.corp.google.com>
- Sho Iwamoto <sho.iwamoto@pd.infn.it>

- Swen Kooij <swenkooij@gmail.com>
- happlebao <c.x.bao@student.ucc.ie>
- icp <pangolin@vivaldi.net>
- serhiy-yevtushenko <syevtushenko@gmail.com>

CHAPTER

SEVEN

LICENSE

```
Copyright (C) 2012-2018 Steven Myint

Permission is hereby granted, free of charge, to any person obtaining a copy of
this software and associated documentation files (the "Software"), to deal in
the Software without restriction, including without limitation the rights to
use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies
of the Software, and to permit persons to whom the Software is furnished to do
so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all
copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
SOFTWARE.
```

CHAPTER

EIGHT

INDICES AND TABLES

- genindex
- modindex
- search