pydocstyle Documentation

Release 6.1.0

Amir Rachum

May 17, 2021

Contents

pydocstyle is a static analysis tool for checking compliance with Python docstring conventions.

pydocstyle supports most of PEP 257 out of the box, but it should not be considered a reference implementation.

pydocstyle supports Python 3.6, 3.7 and 3.8.

CHAPTER 1

Quick Start

1. Install

```
pip install pydocstyle
```

2. Run

```
$ pydocstyle test.py
test.py:18 in private nested class `meta`:
        D101: Docstring missing
test.py:27 in public function `get_user`:
    D300: Use """triple double quotes""" (found '''-quotes)
test:75 in public function `init_database`:
    D201: No blank lines allowed before function docstring (found 1)
...
```

3. Fix your code :)

Contents:

1.1 Usage

1.1.1 Installation

Use pip or easy_install:

```
pip install pydocstyle
```

Alternatively, you can use `pydocstyle.py` source file directly - it is self-contained.

1.1.2 Command Line Interface

Usage

```
Usage: pydocstyle [options] [<file|dir>...]

Options:
  --version             show program's version number and exit
  -h, --help            show this help message and exit
  -e, --explain         show explanation of each error
  -s, --source          show source for each error
  -d, --debug           print debug information
  -v, --verbose         print status information
  --count               print total number of errors to stdout
  --config=<path>       use given config file and disable config discovery
  --match=<pattern>     check only files that exactly match <pattern> regular
                        expression; default is --match='(?!test_).*\.py' which
                        matches files that don't start with 'test_' but end
                        with '.py'
  --match-dir=<pattern>
                        search only dirs that exactly match <pattern> regular
                        expression; default is --match-dir='[^\.].*', which
                        matches all dirs that don't start with a dot
  --ignore-decorators=<decorators>
                        ignore any functions or methods that are decorated by
                        a function with a name fitting the <decorators>
                        regular expression; default is --ignore-decorators=''
                        which does not ignore any decorated functions.

  Note:
    When using --match, --match-dir or --ignore-decorators consider
    whether you should use a single quote (') or a double quote ("),
    depending on your OS, Shell, etc.

  Error Check Options:
    Only one of --select, --ignore or --convention can be specified. If
    none is specified, defaults to `--convention=pep257`. These three
    options select the "basic list" of error codes to check. If you wish
    to change that list (for example, if you selected a known convention
    but wish to ignore a specific error from it or add a new one) you can
    use `--add-[ignore/select]` in order to do so.

    --select=<codes>    choose the basic list of checked errors by specifying
                        which errors to check for (with a list of comma-
                        separated error codes or prefixes). for example:
                        --select=D101,D2
    --ignore=<codes>    choose the basic list of checked errors by specifying
                        which errors to ignore out of all of the available
                        error codes (with a list of comma-separated error
                        codes or prefixes). for example: --ignore=D101,D2
    --convention=<name>
                        choose the basic list of checked errors by specifying
                        an existing convention. Possible conventions: pep257,
                        numpy, google.
    --add-select=<codes>
                        add extra error codes to check to the basic list of
                        errors previously set by --select, --ignore or
                        --convention.
    --add-ignore=<codes>
                        ignore extra error codes by removing them from the
```

```
                        basic list previously set by --select, --ignore or
                        --convention.
```

Note: When using any of the `--select`, `--ignore`, `--add-select`, or `--add-ignore` command line flags, it is possible to pass a prefix for an error code. It will be expanded so that any code beginning with that prefix will match. For example, running the command `pydocstyle --ignore=D4` will ignore all docstring content issues as their error codes beginning with "D4" (i.e. D400, D401, D402, D403, and D404).

Return Code

0	Success - no violations
1	Some code violations were found
2	Illegal usage - see error message

Configuration Files

`pydocstyle` supports *ini*-like and *toml* configuration files. In order for `pydocstyle` to use a configuration file automatically, it must be named one of the following options.

- `setup.cfg`
- `tox.ini`
- `.pydocstyle`
- `.pydocstyle.ini`
- `.pydocstylerc`
- `.pydocstylerc.ini`
- `pyproject.toml`

When searching for a configuration file, `pydocstyle` looks for one of the file specified above *in that exact order*. *ini*-like configuration files must have a `[pydocstyle]` section while *toml* configuration files must have a `[tool.pydocstyle]` section. If a configuration file was not found, `pydocstyle` keeps looking for one up the directory tree until one is found or uses the default configuration.

Note: *toml* configuration file support is only enabled if the `toml` python package is installed. You can ensure that this is the case by installing the `pydocstyle[toml]` optional dependency.

Note: For backwards compatibility purposes, **pydocstyle** supports configuration files named `.pep257`, as well as section header `[pep257]`. However, these are considered deprecated and support will be removed in the next major version.

Available Options

Not all configuration options are available in the configuration files. Available options are:

- `convention`

- `select`
- `ignore`
- `add_select`
- `add_ignore`
- `match`
- `match_dir`
- `ignore_decorators`

See the *Usage* section for more information.

Inheritance

By default, when finding a configuration file, `pydocstyle` tries to inherit the parent directory's configuration and merge them to the local ones.

The merge process is as follows:

- If one of `select`, `ignore` or `convention` was specified in the child configuration - Ignores the parent configuration and set the new error codes to check. Otherwise, simply copies the parent checked error codes.
- If `add-ignore` or `add-select` were specified, adds or removes the specified error codes from the checked error codes list.
- If `match` or `match-dir` were specified - use them. Otherwise, use the parent's.

In order to disable this (useful for configuration files located in your repo's root), simply add `inherit=false` to your configuration file.

Note: If any of `select`, `ignore` or `convention` were specified in the CLI, the configuration files will take no part in choosing which error codes will be checked. `match` and `match-dir` will still take effect.

Example

```
[pydocstyle]
inherit = false
ignore = D100,D203,D405
match = .*\.py
```

In-file configuration

`pydocstyle` supports inline commenting to skip specific checks on specific functions or methods. The supported comments that can be added are:

1. `"# noqa"` skips all checks.
2. `"# noqa: D102,D203"` can be used to skip specific checks. Note that this is compatible with skips from flake8, e.g. `# noqa: D102,E501,D203`.

For example, this will skip the check for a period at the end of a function docstring:

```
>>> def bad_function():  # noqa: D400
...     """Omit a period in the docstring as an exception"""
...     pass
```

1.1.3 Usage with the pre-commit git hooks framework

pydocstyle can be included as a hook for pre-commit. The easiest way to get started is to add this configuration to your `.pre-commit-config.yaml`:

```
-   repo: https://github.com/pycqa/pydocstyle
    rev: 6.1.0  # pick a git hash / tag to point to
    hooks:
    -   id: pydocstyle
```

See the pre-commit docs for how to customize this configuration.

Checked-in python files will be passed as positional arguments so no need to use `--match=*.py`. You can also use command line arguments instead of configuration files to achieve the same effect with less files.

```
- id: pydocstyle
  args:
  - --ignore=D100,D203,D405
  # or multiline
  - |-
          --select=
          D101,
          D2
```

1.2 Error Codes

1.2.1 Grouping

Missing Docstrings	
D100	Missing docstring in public module
D101	Missing docstring in public class
D102	Missing docstring in public method
D103	Missing docstring in public function
D104	Missing docstring in public package
D105	Missing docstring in magic method
D106	Missing docstring in public nested class
D107	Missing docstring in __init__
Whitespace Issues	
D200	One-line docstring should fit on one line with quotes
D201	No blank lines allowed before function docstring
D202	No blank lines allowed after function docstring
D203	1 blank line required before class docstring
D204	1 blank line required after class docstring
D205	1 blank line required between summary line and description
D206	Docstring should be indented with spaces, not tabs
D207	Docstring is under-indented

Continued on next page

Table 1.1 – continued from previous page

D208	Docstring is over-indented
D209	Multi-line docstring closing quotes should be on a separate line
D210	No whitespaces allowed surrounding docstring text
D211	No blank lines allowed before class docstring
D212	Multi-line docstring summary should start at the first line
D213	Multi-line docstring summary should start at the second line
D214	Section is over-indented
D215	Section underline is over-indented
Quotes Issues	
D300	Use “”””triple double quotes”””””
D301	Use r””””” if any backslashes in a docstring
D302	Deprecated: Use u””””” for Unicode docstrings
Docstring Content Issues	
D400	First line should end with a period
D401	First line should be in imperative mood
D401	First line should be in imperative mood; try rephrasing
D402	First line should not be the function’s “signature”
D403	First word of the first line should be properly capitalized
D404	First word of the docstring should not be *This*
D405	Section name should be properly capitalized
D406	Section name should end with a newline
D407	Missing dashed underline after section
D408	Section underline should be in the line following the section’s name
D409	Section underline should match the length of its name
D410	Missing blank line after section
D411	Missing blank line before section
D412	No blank lines allowed between a section header and its content
D413	Missing blank line after last section
D414	Section has no content
D415	First line should end with a period, question mark, or exclamation point
D416	Section name should end with a colon
D417	Missing argument descriptions in the docstring
D418	Function/ Method decorated with @overload shouldn’t contain a docstring

1.2.2 Default conventions

Not all error codes are checked for by default. There are three conventions that may be used by pydocstyle: `pep257`, `numpy` and `google`.

The `pep257` convention (specified in PEP257), which is enabled by default in pydocstyle, checks for all of the above errors except for D203, D212, D213, D214, D215, D404, D405, D406, D407, D408, D409, D410, D411, D413, D415, D416 and D417.

The `numpy` convention added in v2.0.0 supports the numpydoc docstring standard. This checks all of of the errors except for D107, D203, D212, D213, D402, D413, D415, D416, and D417.

The `google` convention added in v4.0.0 supports the Google Python Style Guide. This checks for all the errors except D203, D204, D213, D215, D400, D401, D404, D406, D407, D408, D409 and D413.

These conventions may be specified using *–convention=<name>* when running pydocstyle from the command line or by specifying the convention in a configuration file. See the *Usage* section for more details.

Note: It makes no sense to check the same docstring for both `numpy` and `google` conventions. Therefore, if we

successfully detect that a docstring is in the `numpy` style, we don't check it for `google`.

The reason `numpy` style takes precedence over `google` is that the heuristics of detecting it are better, and we don't want to enforce users to provide external hints to *pydocstyle* in order to let it know which style docstrings are written in.

1.2.3 Publicity

The D1xx group of errors deals with missing docstring in public constructs: modules, classes, methods, etc. It is important to note how publicity is determined and what its effects are.

How publicity is determined

Publicity for all constructs is determined as follows: a construct is considered *public* if -

1. Its immediate parent is public *and*
2. Its name does *not* start with a single or double underscore.
   (a) Note, names that start and end with a double underscore are *public* (e.g. `__init__.py`).

A construct's immediate parent is the construct that contains it. For example, a method's parent is a class object. A class' parent is usually a module, but might also be a function, method, etc. A module can either have no parent, or it can have a parent that is a package.

In order for a construct to be considered public, its immediate parent must also be public. Since this definition is recursive, it means that *all* of its parents need to be public. The corollary is that if a construct is considered private, then all of its descendants are also considered private. For example, a class called `_Foo` is considered private. A method `bar` in `_Foo` is also considered private since its parent is a private class, even though its name does not begin with a single underscore.

Note, a module's parent is recursively checked upward until we reach a directory in `sys.path` to avoid considering the complete filepath of a module. For example, consider the module `/_foo/bar/baz.py`. If `PYTHONPATH` is set to `/`, then `baz.py` is *private*. If `PYTHONPATH` is set to `/_foo/`, then `baz.py` is *public*.

Modules are parsed to look if `__all__` is defined. If so, only those top level constructs are considered public. The parser looks for `__all__` defined as a literal list or tuple. As the parser doesn't execute the module, any mutation of `__all__` will not be considered.

How publicity affects error reports

The immediate effect of a construct being determined as private is that no D1xx errors will be reported for it (or its children, as the previous section explains). A private method, for instance, will not generate a D102 error, even if it has no docstring.

However, it is important to note that while docstring are optional for private construct, they are still required to adhere to your style guide. So if a private module *_foo.py* does not have a docstring, it will not generate a D100 error, but if it *does* have a docstring, that docstring might generate other errors.

1.3 Release Notes

pydocstyle version numbers follow the Semantic Versioning specification.

1.3.1 6.1.0 - May 17th, 2021

New Features

- Enable full toml configuration and pyproject.toml (#534).

1.3.2 6.0.0 - March 18th, 2021

Major Updates

- Support for Python 3.5 has been dropped (#510).

New Features

- Add flag to disable *# noqa* comment processing in API (#485).
- Methods, Functions and Nested functions that have a docstring now throw D418 (#511).
- Methods decorated with @overload no longer reported as D102 (#511).
- Functions and nested functions decorated with @overload no longer reported as D103 (#511).

Bug Fixes

- Treat "package" as an imperative verb for D401 (#356).
- Fix the parsing of decorated one line functions (#499).

1.3.3 5.1.2 - September 13th, 2020

New Features

- Methods, Functions and Nested functions that have a docstring now throw D418 (#511).
- Methods decorated with @overload no longer reported as D102.
- Functions and nested functions decorated with @overload no longer reported as D103.

1.3.4 5.1.1 - August 29th, 2020

Bug Fixes

- Fix `IndexError` crash on one-line backslashed docstrings (#506).

1.3.5 5.1.0 - August 22nd, 2020

New Features

- Skip function arguments prefixed with _ in D417 check (#440).

Bug Fixes

- Update convention support documentation (#386, #393)
- Detect inner asynchronous functions for D202 (#467)
- Fix indentation error while parsing class methods (#441).
- Fix a bug in parsing Google-style argument description. The bug caused some argument names to go unreported in D417 (#448).

- Fixed an issue where skipping errors on module level docstring via #noqa failed when there where more prior comments (#446).
- Support backslash-continued descriptions in docstrings (#472).
- Correctly detect publicity of modules inside directories (#470, #494).

1.3.6 5.0.2 - January 8th, 2020

Bug Fixes

- Fix `DeprecationWarning` / `SyntaxError` "invalid escape sequence" with Python 3.6+ (#445).

1.3.7 5.0.1 - December 9th, 2019

Bug Fixes

- Fixed an issue where AttributeError was raised when parsing the parameter section of a class docstring (#434, #436).

1.3.8 5.0.0 - December 9th, 2019

Major Updates

- Support for Python 3.4 has been dropped (#402).

New Features

- Extend support for detecting missing arguments in Google style docstrings to method calls (#384).
- Extend support for detecting missing argument description in Numpy style docstrings (#407).
- Added support for Python 3.8 (#423).
- Allow skipping errors on module level docstring via #noqa (#427).
- Whitespace is ignored with set options split across multiple lines (#221).

Bug Fixes

- Remove D413 from the google convention (#430).
- Remove D413 from the pep257 convention (#404).
- Replace *semicolon* with *colon* in D416 messages. (#409)
- D301 (Use r""" if any backslashes in a docstring) does not trigger on backslashes for line continuation or unicode literals `\u...` and `\N...` anymore. These are considered intended elements of the docstring and thus should not be escaped by using a raw docstring (#365).
- Fix decorator parsing (#411).
- Google-style sections no longer cause false errors when used with Numpy-style sections (#388, #424).
- D202: Allow a blank line after function docstring when followed by declaration of an inner function or class (#395, #426).
- Fix D401 and D404 checks not working for docstrings containing only one word and ending with non-alpha character (#421)

1.3.9 4.0.1 - August 14th, 2019

Bug Fixes

- D401: Fixed a false positive where one stem had multiple imperative forms, e.g., init and initialize / initiate (#382).
- Fix parser hanging when there's a comment directly after `__all__` (#391, #366).
- Fixed RST error in table which resulted in the online documentation missing the violation code table (#396).
- Fixed IndentationError when parsing function arguments (#392).

1.3.10 4.0.0 - July 6th, 2019

Major Updates

- Support for Python 2.x and PyPy has been dropped (#340).
- Added initial support for Google convention (#357).

New Features

- Added pre-commit hook (#346)

Bug Fixes

- Fix parsing tuple syntax `__all__` (#355, #352).

1.3.11 3.0.0 - October 14th, 2018

Major Updates

- Support for Python 3.3 has been dropped (#315, #316).
- Added support for Python 3.7 (#324).

New features

- Violations are now reported on the line where the docstring starts, not the line of the `def`/`class` it corresponds to (#238, #83).
- Updated description of pep257 and numpy conventions (#300).
- `__all__` parsing is now done on a best-effort basis - if `__all__` can't be statically determined, it will be ignored (#320, #313).

Bug Fixes

- Fixed a false-positive recognition of section names causing D405 to be reported (#311, #317).
- Fixed a bug where functions that don't end with a newline will sometimes raise an exception (#321, #336).

1.3.12 2.1.1 - October 9th, 2017

Bug Fixes

- Changed wheel configuration to be NOT universal, as #281 added `configparser` as a dependency for Python 2.7.
- Updated usage documentation.

1.3.13 2.1.0 - October 8th, 2017

New Features

- Public nested classes missing a docstring are now reported as D106 instead of D101 (#198, #261).
- `__init__` methods missing a docstring are now reported as D107 instead of D102 (#273, #277).
- Added support for Python 3.6 (#270).
- Specifying an invalid error code prefix (e.g., `--select=D9`) will print a warning message to `stderr` (#253, #279).
- Configuration files now support multiple-lined entries (#250, #281).
- Improved description of how error selection works in the help section (#231, #283).

Bug Fixes

- Fixed an issue where the `--source` flag would result in improperly spaced output (#256, #257, #260).
- Fixed an issue where if a first word in a docstring had Unicode characters and the docstring was not a unicode string, an exception would be raised (#258, #264).
- Configuration files that were specified by CLI and don't contain a valid section name will now issue a warning to `stderr` (#276, #280).
- Removed D107 from the numpy convention (#288).

1.3.14 2.0.0 - April 18th, 2017

Major Updates

- Support for `numpy` conventions verification has been added (#129, #226).
- Support for Python 2.6 has been dropped (#206, #217).
- Support for PyPy3 has been temporarily dropped, until it will be equivalent to CPython 3.3+ and supported by `pip` (#223).
- Support for the `pep257` console script has been dropped. Only the `pydocstyle` console script should be used (#216, #218).
- Errors are now printed to `stdout` instead of `stderr` (#201, #210).

New Features

- Decorator-based skipping via `--ignore-decorators` has been added (#204).
- Support for using pycodestyle style wildcards has been added (#72, #209).
- Superfluous opening quotes are now reported as part of D300 (#166, #225).
- Fixed a false-positive recognition of *D410* and added *D412* (#230, #233).
- Added `--config=<path>` flag to override the normal config file discovery and choose a specific config file (#117, #247).
- Support for specifying error codes with partial prefix has been added, e.g., `--select=D101,D2` (#72, #209).
- All configuration file can now have the `.ini` extension (#237).
- Added better imperative mood checks using third party stemmer (#235, #68).

Bug Fixes

- Made parser more robust to bad source files (#168, #214)
- Modules are now considered private if their name starts with a single underscore. This is a bugfix where "public module" (D100) was reported regardless of module name (#199, #222).
- Removed error when `__all__` is a list (#62, #227).
- Fixed a bug where the `@` sign was used as a matrix multiplication operator in Python 3.5, but was considered a decorator by the parser (#246, #191).

1.3.15 1.1.1 - October 4th, 2016

Bug Fixes

- Fixed an issue where the `flake8-docstrings` failed when accessing some public API from `pydocstyle`.

1.3.16 1.1.0 - September 29th, 2016

Major Updates

- `pydocstyle` is no longer a single file. This might make it difficult for some users to just add it to their project, but the project has reached certain complexity where splitting it into modules was necessary (#200).

New Features

- Added the optional error codes D212 and D213, for checking whether the summary of a multi-line docstring starts at the first line, respectively at the second line (#174).
- Added D404 - First word of the docstring should not be "This". It is turned off by default (#183).
- Added the ability to ignore specific function and method docstrings with inline comments:
  1. "# noqa" skips all checks.
  2. "# noqa: D102,D203" can be used to skip specific checks.

Bug Fixes

- Fixed an issue where file paths were printed in lower case (#179, #181).
- The error code D300 is now also being reported if a docstring has uppercase literals (`R` or `U`) as prefix (#176).
- Fixed a bug where an `__all__` error was reported when `__all__` was imported from another module with a different name (#182, #187).
- Fixed a bug where `raise X from Y` syntax caused `pydocstyle` to crash (#196, #200).

1.3.17 1.0.0 - January 30th, 2016

Major Updates

- The project was renamed to **pydocstyle** and the new release will be 1.0.0!

New Features

- Added support for Python 3.5 (#145).
- Classes nested inside classes are no longer considered private. Nested classes are considered public if their names are not prepended with an underscore and if their parent class is public, recursively (#13, #146).
- Added the D403 error code - "First word of the first line should be properly capitalized". This new error is turned on by default (#164, #165, #170).

- Added support for `.pydocstylerc` and as configuration file name (#140, #173).

Bug Fixes

- Fixed an issue where a `NameError` was raised when parsing complex definitions of `__all__` (#142, #143).
- Fixed a bug where D202 was falsely reported when a function with just a docstring and no content was followed by a comment (#165).
- Fixed wrong `__all__` definition in main module (#150, #156).
- Fixed a bug where an `AssertionError` could occur when parsing `__future__` imports (#154).

1.4 Older Versions

Note: Versions documented below are before renaming the project from **pep257** to **pydocstyle**.

1.4.1 0.7.0 - October 9th, 2015

New Features

- Added the D104 error code - "Missing docstring in public package". This new error is turned on by default. Missing docstring in `__init__.py` files which previously resulted in D100 errors ("Missing docstring in public module") will now result in D104 (#105, #127).
- Added the D105 error code - "Missing docstring in magic method'. This new error is turned on by default. Missing docstrings in magic method which previously resulted in D102 error ("Missing docstring in public method") will now result in D105. Note that exceptions to this rule are variadic magic methods - specifically `__init__`, `__call__` and `__new__`, which will be considered non-magic and missing docstrings in them will result in D102 (#60, #139).
- Support the option to exclude all error codes. Running pep257 with `--select=` (or `select=` in the configuration file) will exclude all errors which could then be added one by one using `add-select`. Useful for projects new to pep257 (#132, #135).
- Added check D211: No blank lines allowed before class docstring. This change is a result of a change to the official PEP257 convention. Therefore, D211 will now be checked by default instead of D203, which required a single blank line before a class docstring (#137).
- Configuration files are now handled correctly. The closer a configuration file is to a checked file the more it matters. Configuration files no longer support `explain`, `source`, `debug`, `verbose` or `count` (#133).

Bug Fixes

- On Python 2.x, D302 ("Use u""" for Unicode docstrings") is not reported if *unicode_literals* is imported from *__future__* (#113, #134).
- Fixed a bug where there was no executable for *pep257* on Windows (#73, #136).

1.4.2 0.6.0 - July 20th, 2015

New Features

- Added support for more flexible error selections using `--ignore`, `--select`, `--convention`, `--add-ignore` and `--add-select` (#96, #123).

Bug Fixes

- Property setter and deleter methods are now treated as private and do not require docstrings separate from the main property method (#69, #107).
- Fixed an issue where pep257 did not accept docstrings that are both unicode and raw in Python 2.x (#116, #119).
- Fixed an issue where Python 3.x files with Unicode encodings were not read correctly (#118).

1.4.3 0.5.0 - March 14th, 2015

New Features

- Added check D210: No whitespaces allowed surrounding docstring text (#95).
- Added real documentation rendering using Sphinx (#100, #101).

Bug Fixes

- Removed log level configuration from module level (#98).
- D205 used to check that there was *a* blank line between the one line summary and the description. It now checks that there is *exactly* one blank line between them (#79).
- Fixed a bug where `--match-dir` was not properly respected (#108, #109).

1.4.4 0.4.1 - January 10th, 2015

Bug Fixes

- Getting `ImportError` when trying to run pep257 as the installed script (#92, #93).

1.4.5 0.4.0 - January 4th, 2015

> **Warning:** A fatal bug was discovered in this version (#92). Please use a newer version.

New Features

- Added configuration file support (#58, #87).
- Added a `--count` flag that prints the number of violations found (#86, #89).
- Added support for Python 3.4, PyPy and PyPy3 (#81).

Bug Fixes

- Fixed broken tests (#74).
- Fixed parsing various colon and parenthesis combinations in definitions (#82).
- Allow for greater flexibility in parsing `__all__` (#67).
- Fixed handling of one-liner definitions (#77).

1.4.6 0.3.2 - March 11th, 2014

First documented release!

1.5 License

CHAPTER 2

Credits

pydocstyle is a rename and continuation of pep257, a project created by Vladimir Keleshev.

Maintained by Amir Rachum and Sambhav Kothari.