Documentation for Black

Release 18.3a4

Łukasz Langa and contributors to Black

Mar 30, 2018

Contents

1 Testimonials **3**

2 Contents **5**

2.1 Installation and Usage 5
2.2 The *Black* code style 6
2.3 Editor integration 8
2.4 Contributing to Black 9
2.5 Change Log 10
2.6 Developer reference 12
2.7 Authors 13

3 Indices and tables **15**

By using *Black*, you agree to cease control over minutiae of hand-formatting. In return, *Black* gives you speed, determinism, and freedom from *pycodestyle* nagging about formatting. You will save time and mental energy for more important matters.

Black makes code review faster by producing the smallest diffs possible. Blackened code looks the same regardless of the project you're reading. Formatting becomes transparent after a while and you can focus on the content instead.

Note: Black is an early pre-release.

CHAPTER 1

Testimonials

Dusty Phillips, writer:

> *Black is opinionated so you don't have to be.*

Hynek Schlawack, creator of attrs, core developer of Twisted and CPython:

> *An auto-formatter that doesn't suck is all I want for Xmas!*

Carl Meyer, Django core developer:

> *At least the name is good.*

Kenneth Reitz, creator of requests and pipenv:

> *This vastly improves the formatting of our code. Thanks a ton!*

CHAPTER 2

Contents

2.1 Installation and Usage

2.1.1 Installation

Black can be installed by running `pip install black`. It requires Python 3.6.0+ to run but you can reformat Python 2 code with it, too.

2.1.2 Usage

To get started right away with sensible defaults:

```
black {source_file_or_directory}
```

2.1.3 Command line options

Black doesn't provide many options. You can list them by running `black --help`:

```
black [OPTIONS] [SRC]...

Options:
  -l, --line-length INTEGER   Where to wrap around.  [default: 88]
  --check                     Don't write back the files, just return the
                              status.  Return code 0 means nothing would
                              change.  Return code 1 means some files would be
                              reformatted.  Return code 123 means there was an
                              internal error.
  --fast / --safe             If --fast given, skip temporary sanity checks.
                              [default: --safe]
  --version                   Show the version and exit.
  --help                      Show this message and exit.
```

Black is a well-behaved Unix-style command-line tool:

- it does nothing if no sources are passed to it;
- it will read from standard input and write to standard output if `-` is used as the filename;
- it only outputs messages to users on standard error;
- exits with code 0 unless an internal error occured (or `--check` was used).

2.1.4 NOTE: This is an early pre-release

Black can already successfully format itself and the standard library. It also sports a decent test suite. However, it is still very new. Things will probably be wonky for a while. This is made explicit by the "Alpha" trove classifier, as well as by the "a" in the version number. What this means for you is that **until the formatter becomes stable, you should expect some formatting to change in the future**.

Also, as a temporary safety measure, *Black* will check that the reformatted code still produces a valid AST that is equivalent to the original. This slows it down. If you're feeling confident, use `--fast`.

2.2 The *Black* code style

Black reformats entire files in place. It is not configurable. It doesn't take previous formatting into account. It doesn't reformat blocks that start with `# fmt: off` and end with `# fmt: on`. It also recognizes YAPF's block comments to the same effect, as a courtesy for straddling code.

2.2.1 How *Black* wraps lines

Black ignores previous formatting and applies uniform horizontal and vertical whitespace to your code. The rules for horizontal whitespace are pretty obvious and can be summarized as: do whatever makes `pycodestyle` happy. The coding style used by *Black* can be viewed as a strict subset of PEP 8.

As for vertical whitespace, *Black* tries to render one full expression or simple statement per line. If this fits the allotted line length, great.

```
# in:

l = [1,
     2,
     3,
]

# out:

l = [1, 2, 3]
```

If not, *Black* will look at the contents of the first outer matching brackets and put that in a separate indented line.

```
# in:

l = [[n for n in list_bosses()], [n for n in list_employees()]]

# out:

```

(continues on next page)

(continued from previous page)

```
l = [
    [n for n in list_bosses()], [n for n in list_employees()]
]
```

If that still doesn't fit the bill, it will decompose the internal expression further using the same rule, indenting matching brackets every time. If the contents of the matching brackets pair are comma-separated (like an argument list, or a dict literal, and so on) then *Black* will first try to keep them on the same line with the matching brackets. If that doesn't work, it will put all of them in separate lines.

```
# in:

def very_important_function(template: str, *variables, file: os.PathLike, debug: bool␣
↪= False):
    """Applies `variables` to the `template` and writes to `file`."""
    with open(file, 'w') as f:
        ...

# out:

def very_important_function(
    template: str,
    *variables,
    file: os.PathLike,
    debug: bool = False,
):
    """Applies `variables` to the `template` and writes to `file`."""
    with open(file, 'w') as f:
        ...
```

You might have noticed that closing brackets are always dedented and that a trailing comma is always added. Such formatting produces smaller diffs; when you add or remove an element, it's always just one line. Also, having the closing bracket dedented provides a clear delimiter between two distinct sections of the code that otherwise share the same indentation level (like the arguments list and the docstring in the example above).

2.2.2 Line length

You probably noticed the peculiar default line length. *Black* defaults to 88 characters per line, which happens to be 10% over 80. This number was found to produce significantly shorter files than sticking with 80 (the most popular), or even 79 (used by the standard library). In general, 90-ish seems like the wise choice.

If you're paid by the line of code you write, you can pass `--line-length` with a lower number. *Black* will try to respect that. However, sometimes it won't be able to without breaking other rules. In those rare cases, auto-formatted code will exceed your allotted limit.

You can also increase it, but remember that people with sight disabilities find it harder to work with line lengths exceeding 100 characters. It also adversely affects side-by-side diff review on typical screen resolutions. Long lines also make it harder to present code neatly in documentation or talk slides.

If you're using Flake8, you can bump `max-line-length` to 88 and forget about it. Alternatively, use Bugbear's B950 warning instead of E501 and keep the max line length at 80 which you are probably already using. You'd do it like this:

```
[flake8]
max-line-length = 80
...
```

(continues on next page)

(continued from previous page)

```
select = C,E,F,W,B,B950
ignore = E501
```

You'll find *Black*'s own .flake8 config file is configured like this. If you're curious about the reasoning behind B950, Bugbear's documentation explains it. The tl;dr is "it's like highway speed limits, we won't bother you if you overdo it by a few km/h".

2.2.3 Empty lines

Black avoids spurious vertical whitespace. This is in the spirit of PEP 8 which says that in-function vertical whitespace should only be used sparingly. One exception is control flow statements: *Black* will always emit an extra empty line after `return`, `raise`, `break`, `continue`, and `yield`. This is to make changes in control flow more prominent to readers of your code.

Black will allow single empty lines inside functions, and single and double empty lines on module level left by the original editors, except when they're within parenthesized expressions. Since such expressions are always reformatted to fit minimal space, this whitespace is lost.

It will also insert proper spacing before and after function definitions. It's one line before and after inner functions and two lines before and after module-level functions. *Black* will put those empty lines also between the function definition and any standalone comments that immediately precede the given function. If you want to comment on the entire function, use a docstring or put a leading comment in the function body.

2.2.4 Trailing commas

Black will add trailing commas to expressions that are split by comma where each element is on its own line. This includes function signatures.

Unnecessary trailing commas are removed if an expression fits in one line. This makes it 1% more likely that your line won't exceed the allotted line length limit. Moreover, in this scenario, if you added another argument to your call, you'd probably fit it in the same line anyway. That doesn't make diffs any larger.

One exception to removing trailing commas is tuple expressions with just one element. In this case *Black* won't touch the single trailing comma as this would unexpectedly change the underlying data type. Note that this is also the case when commas are used while indexing. This is a tuple in disguise: `numpy_array[3, ]`.

One exception to adding trailing commas is function signatures containing `*`, `*args`, or `**kwargs`. In this case a trailing comma is only safe to use on Python 3.6. *Black* will detect if your file is already 3.6+ only and use trailing commas in this situation. If you wonder how it knows, it looks for f-strings and existing use of trailing commas in function signatures that have stars in them. In other words, if you'd like a trailing comma in this situation and *Black* didn't recognize it was safe to do so, put it there manually and *Black* will keep it.

2.3 Editor integration

2.3.1 Emacs

Use proofit404/blacken.

2.3.2 Vim

Commands and shortcuts:

- `,=` or `:Black` to format the entire file (ranges not supported);
- `:BlackUpgrade` to upgrade *Black* inside the virtualenv;
- `:BlackVersion` to get the current version of *Black* inside the virtualenv.

Configuration:

- `g:black_fast` (defaults to `0`)
- `g:black_linelength` (defaults to `88`)
- `g:black_virtualenv` (defaults to `~/.vim/black`)

To install, copy the plugin from vim/plugin/black.vim. Let me know if this requires any changes to work with Vim 8's builtin `packadd`, or Pathogen, or Vundle, and so on.

This plugin **requires Vim 7.0+ built with Python 3.6+ support**. It needs Python 3.6 to be able to run *Black* inside the Vim process which is much faster than calling an external command.

On first run, the plugin creates its own virtualenv using the right Python version and automatically installs *Black*. You can upgrade it later by calling `:BlackUpgrade` and restarting Vim.

If you need to do anything special to make your virtualenv work and install *Black* (for example you want to run a version from master), just create a virtualenv manually and point `g:black_virtualenv` to it. The plugin will use it.

How to get Vim with Python 3.6? On Ubuntu 17.10 Vim comes with Python 3.6 by default. On macOS with HomeBrew run: `brew install vim --with-python3`. When building Vim from source, use: `./configure --enable-python3interp=yes`. There's many guides online how to do this.

2.3.3 Visual Studio Code

Use joslarson.black-vscode.

2.3.4 Other editors

Atom/Nuclide integration is planned by the author, others will require external contributions.

Patches welcome!

Any tool that can pipe code through *Black* using its stdio mode (just use `-` as the file name). The formatted code will be returned on stdout (unless `--check` was passed). *Black* will still emit messages on stderr but that shouldn't affect your use case.

This can be used for example with PyCharm's File Watchers.

2.4 Contributing to Black

Welcome! Happy to see you willing to make the project better. Have you read the entire user documentation yet?

2.4.1 Bird's eye view

In terms of inspiration, *Black* is about as configurable as *gofmt* and *rustfmt* are. This is deliberate.

Bug reports and fixes are always welcome! Please follow the issue template on GitHub for best results.

Before you suggest a new feature or configuration knob, ask yourself why you want it. If it enables better integration with some workflow, fixes an inconsistency, speeds things up, and so on - go for it! On the other hand, if your answer is "because I don't like a particular formatting" then you're not ready to embrace *Black* yet. Such changes are unlikely to get accepted. You can still try but prepare to be disappointed.

2.4.2 Technicalities

Development on the latest version of Python is preferred. As of this writing it's 3.6.4. You can use any operating system. I am using macOS myself and CentOS at work.

Install all development dependencies using:

```
$ pipenv install --dev
```

If you haven't used `pipenv` before but are comfortable with virtualenvs, just run `pip install pipenv` in the virtualenv you're already using and invoke the command above from the cloned Black repo. It will do the correct thing.

Before submitting pull requests, run tests with:

```
$ python setup.py test
```

Also run mypy and flake8 on `black.py` and `test_black.py`. Travis will run all that for you but if you left any errors here, it will be quicker and less embarrassing to fix them locally ;-)

2.4.3 Hygiene

If you're fixing a bug, add a test. Run it first to confirm it fails, then fix the bug, run it again to confirm it's really fixed.

If adding a new feature, add a test. In fact, always add a test. But wait, before adding any large feature, first open an issue for us to discuss the idea first.

2.4.4 Finally

Thanks again for your interest in improving the project! You're taking action when most people decide to sit and watch.

2.5 Change Log

2.5.1 18.3a5 (unreleased)

- fixed 18.3a4 regression: don't crash and burn on empty lines with trailing whitespace (#80)
- only allow up to two empty lines on module level and only single empty lines within functions (#74)

2.5.2 18.3a4

- `# fmt: off` and `# fmt: on` are implemented (#5)
- automatic detection of deprecated Python 2 forms of print statements and exec statements in the formatted file (#49)

- use proper spaces for complex expressions in default values of typed function arguments (#60)
- only return exit code 1 when –check is used (#50)
- don't remove single trailing commas from square bracket indexing (#59)
- don't omit whitespace if the previous factor leaf wasn't a math operator (#55)
- omit extra space in kwarg unpacking if it's the first argument (#46)
- omit extra space in Sphinx auto-attribute comments (#68)

2.5.3 18.3a3

- don't remove single empty lines outside of bracketed expressions (#19)
- added ability to pipe formatting from stdin to stdin (#25)
- restored ability to format code with legacy usage of `async` as a name (#20, #42)
- even better handling of numpy-style array indexing (#33, again)

2.5.4 18.3a2

- changed positioning of binary operators to occur at beginning of lines instead of at the end, following a recent change to PEP8 (#21)
- ignore empty bracket pairs while splitting. This avoids very weirdly looking formattings (#34, #35)
- remove a trailing comma if there is a single argument to a call
- if top level functions were separated by a comment, don't put four empty lines after the upper function
- fixed unstable formatting of newlines with imports
- fixed unintentional folding of post scriptum standalone comments into last statement if it was a simple statement (#18, #28)
- fixed missing space in numpy-style array indexing (#33)
- fixed spurious space after star-based unary expressions (#31)

2.5.5 18.3a1

- added `--check`
- only put trailing commas in function signatures and calls if it's safe to do so. If the file is Python 3.6+ it's always safe, otherwise only safe if there are no `*args` or `**kwargs` used in the signature or call. (#8)
- fixed invalid spacing of dots in relative imports (#6, #13)
- fixed invalid splitting after comma on unpacked variables in for-loops (#23)
- fixed spurious space in parenthesized set expressions (#7)
- fixed spurious space after opening parentheses and in default arguments (#14, #17)
- fixed spurious space after unary operators when the operand was a complex expression (#15)

2.5.6 18.3a0

- first published version, Happy Day 2018!
- alpha quality
- date-versioned (see: https://calver.org/)

2.6 Developer reference

Contents are subject to change.

2.6.1 *Black* classes

Contents are subject to change.

BracketTracker

EmptyLineTracker

Line

LineGenerator

Report

UnformattedLines

Visitor

2.6.2 *Black* functions

Contents are subject to change.

Assertions and checks

Formatting

File operations

Parsing

Split functions

Utilities

black.DebugVisitor.**show**(*code: str*) → None
Pretty-print the lib2to3 AST of a given string of *code*.

2.6.3 *Black* exceptions

Contents are subject to change.

2.7 Authors

Glued together by Łukasz Langa.

Maintained with Carol Willing and Carl Meyer.

Multiple contributions by:

- Artem Malyshev
- Daniel M. Capella
- Eli Treuherz
- Hugo van Kemenade
- Mika
- Osaetin Daniel

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

Index

B

black.DebugVisitor.show() (in module black), 12