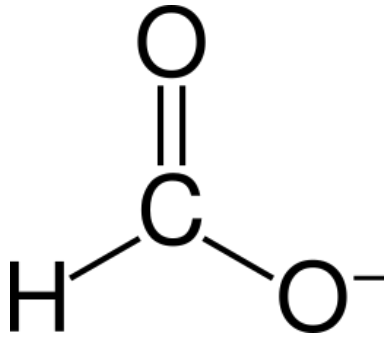

formate

Release 0.7.0

Python formatting mate.

Dominic Davis-Foster

Oct 20, 2023

Contents

Chapter

ONE

Definitions

for·mate

1.1 Etymology 1

formic + -ate

formate (plural formates)

n. (organic chemistry) Any salt or ester of formic acid.

1.2 Etymology 2

Portmanteau of format + mate

formate

n. (programming) A Python autoformatting tool.

Chapter

TWO

Installation

2.1 from PyPI

```
$ python3 -m pip install formate --user
```

2.2 from Anaconda

First add the required channels

```
$ conda config --add channels https://conda.anaconda.org/conda-forge
$ conda config --add channels https://conda.anaconda.org/domdfcoding
```

Then install

```
$ conda install formate
```

2.3 from GitHub

```
$ python3 -m pip install git+https://github.com/python-formate/formate@master --user
```

Part I

Documentation

Chapter

THREE

Usage

`formate` runs a series of user-selected hooks which reformat Python source files. This can include *changing quote characters*, *rewriting imports*, and calling tools such as isort and yapf.

3.1 Command Line

Reformat the given Python source files.

```
formate [OPTIONS] [FILENAME]...
```

Options

-c, --config-file <config_file>
: The path or filename of the TOML configuration file to use. If a filename is given it is searched for in the current and parent directories.

  Default formate.toml

-e, --exclude <PATTERN>
: Patterns for files to exclude from formatting.

-v, --verbose
: Show verbose output.

--colour, --no-colour
: Whether to use coloured output.

-T, --traceback
: Show the complete traceback on error.

--diff
: Show a diff of changes made

--version
: Show the version and exit.

Arguments

FILENAME
Optional argument(s). Default `None`

3.2 As a `pre-commit` hook

`formate` can also be used as a pre-commit hook. To do so, add the following to your .pre-commit-config.yaml file:

```
- repo: https://github.com/python-formate/formate
  rev: 0.7.0
  hooks:
  - id: formate
    args:
    - --verbose
```

The `args` option can be used to provide the command line arguments shown above. By default `formate` is run with `--verbose --diff`

Chapter

FOUR

Configuration

`formate` is configured using the `formate.toml` file in the root of your project (alongside `setup.py`, `tox.ini` etc.). The file uses the TOML syntax, built around a top-level mapping of two keys. The *`-c / --config-file`* option can be used to point to a file with a different name, but at this time the file must be TOML and must have the same layout.

The two top-level keys are *hooks* and *config*.

4.1 `hooks`

This is a mapping of hook names to their settings. The values can be either:

- an integer, representing the priority of the hook.
- a mapping of the following:
  - `priority` – an integer, representing the priority of the hook.
  - `args` – a list of positional arguments to pass to the hook function. Optional. Default `()`.
  - `kwargs` – a mapping of keyword arguments to pass to the hook. Optional. Default `{}`.

The TOML syntax allows for the mapping to spread over multiple tables, like so:

```
[hooks]
reformat-generics = 40
collections-import-rewrite = 20
noqa-reformat = 60
ellipsis-reformat = 70

[hooks.isort]
priority = 50

[hooks.isort.kwargs]
multi_line_output = 8
use_parentheses = true
remove_redundant_aliases = true
```

4.2 config

This is a mapping of general configuration settings, which hooks can use as they please. Common keys include `indent`, indicating the type of indent to use, and `line_length`, indicating how long lines may be.

Example:

```
[config]
indent = "\t"
line_length = 115
```

Alternatively the configuration may be placed in the `pyproject.toml` file defined in **PEP 518**. The only difference is that `hooks` and `config` become `tool.formate.hooks` and `tool.formate.config`. You must also pass *`--config-file pyproject.toml`* on the command line.

Chapter

FIVE

Built-in Hooks

`formate` ships with several hooks out of the box:

5.1 dynamic_quotes

Reformats quotes in the given source, and returns the reformatted source.

This hook takes no arguments.

5.2 collections-import-rewrite

Identify deprecated `from collections import <abc>` imports, and rewrite them as `from collections.abc import <abc>`.

This hook takes no arguments.

5.3 noqa_reformat

Pull `# noqa: ...` comments that immediately follow docstrings back up to the end of the correct line.

This hook takes no arguments.

5.4 check_ast

Check the source can be parsed as a Python Abstract Syntax Tree. This could be called early in the execution – to check the file is valid before starting reformatting – and again at the end to ensure no errors were introduced by the reformatting.

This hook takes no arguments.

5.5 squish_stubs

Squash type stubs by removing unnecessary blank lines.

This hook takes no arguments.

5.6 ellipsis_reformat

Move ellipses (`...`) for type stubs onto the end of the stub definition.

Before:

```
def foo(value: str) -> int:
    ...
```

After:

```
def foo(value: str) -> int: ...
```

This hook takes no arguments.

5.7 reformat-generics

Reformats generics (`typing.Generic`, `typing.Union`, `typing.Callable` etc.).

This hook takes a single keyword argument: `indent`. The indent can also be configured via the `indent` key in the *config* table.

5.8 isort

Calls isort, using the given keyword arguments as its configuration.

This hook only takes keyword arguments.

The max line length can be provided via the `line_length` keyword argument or in the *config* table as `line_length`.

5.9 yapf

Calls yapf, using the given keyword arguments as its configuration.

This hook only takes keyword arguments.

The indent can be configured via the `use_tabs` keyword argument or in the *config* table as `indent`.

Chapter

SIX

Creating your own hooks

It is easy to create your own hooks to extend `formate`. A basic hook looks like this:

```
def make_upper(source: str) -> str:
    """
    Make all the source uppercase.

    :param source: The source to reformat.

    :return: The reformatted source.
    """

    return source.upper()
```

An entry point must be configured for the hook. For setuptools:

```
[options.entry_points]
formate_hooks =
    make_upper=<import path>:make_upper
```

or in `pyproject.toml` with **PEP 621**:

```
[project.entry-points.formate_hooks]
make_upper = "<import path>:make_upper"
```

Hooks may also accept positional and/or keyword arguments, either named or with `*args` and `**kwargs`:

```
def change_case(source: str, upper: bool = True) -> str:
    """
    Change the case of the source.

    :param source: The source to reformat.
    :param upper: Makes the source uppercase.

    :return: The reformatted source.
    """

    if upper:
        return source.upper()
    else:
        return source.lower()
```

Some hooks may require access the the global configuration dict (the *[config]* table in `formate.toml`). Hooks can request this by using the *@formate.config.wants_global_config* decorator, which provides the configuration as the `formate_global_config` keyword argument:

```
def change_indents(
        source: str,
        formate_global_config: Optional[Mapping] = None,
        ) -> str:
    """
    Change the indents of the source.

    :param source: The source to reformat.
    :param formate_global_config: The global configuration dictionary. Optional.

    :return: The reformatted source.
    """

    if formate_global_config is None:
        formate_global_config = {}

    indent = formate_global_config.get("indent", '\t')

    return re.sub("(    |\t)", indent, source)
```

Similarly, some hooks may want to know which filename is being reformatted. They can request this using the *@formate.config.wants_filename* decorator (new in version 0.2.0), which provides the configuration as the `formate_filename` keyword argument:

```
def lint_stubs(source: str, formate_filename: PathLike) -> str:
    """
    Lint Python stub files.

    :param source: The source to check.
    :param formate_filename: The name of the source file,
        to ensure this hook only runs on type stubs.

    :return: The reformatted source.
    """

    if os.path.splitext(formate_filename)[1] != ".pyi":
        return source

    ...

    return reformatted_source
```

See repo-helper/formate-black for an example extension.

Chapter

SEVEN

Changelog

0.4.9

Bugs Fixed

- *formate.dynamic_quotes()* – Preserve surrogates in strings. This prevents a crash when attempting to write the resulting file.

0.4.8

Bugs Fixed

- *formate.reformat_file()* – Only write to the file if there have been any changes. This avoids unnecessary changes to the mtime.
- *formate.reformat_generics* – Don't crash if a generic's name contains a `..`

0.4.7

Bugs Fixed

- *formate.reformat_generics* – Correctly handle boolean values in Literals.

0.4.6

Bugs Fixed

- *formate.dynamic_quotes* – Preserve quote style in docstrings.

0.4.5

Bugs Fixed

- *formate.config* – The decorators now use a type variable to indicate to type checkers the returned object has the same type as the decorated object.
- *formate.isort_hook()* and *formate.yapf_hook()* – Don't crash when keys are missing from `formate_global_config` and aren't in `**kwargs`.

0.4.4

Enhancements

- Switched to `dom_toml` for reading TOML files.
- Relaxed the yapf version requirement to allow 0.31.0 in addition to 0.30.0
- Relaxed the isort version requirement from `isort<=5.6.4,>=5.5.2` to `isort<=5.9.0,>=5.5.2`

0.4.3

- Switched to whey as the build backend.

0.4.2

Bugs Fixed

- *formate.mini_hooks.squish_stubs()* – Ensure space between classes and functions is preserved in cases where there would be no space between the class and a method.

0.4.1

Bugs Fixed

- *formate.mini_hooks.squish_stubs()* – Don't crash due to accessing an out-of-range value from a list.

0.4.0

Enhancements

- *formate.mini_hooks.squish_stubs()* – Remove whitespace between the class definition and first single-line function.

0.3.2

Bugs Fixed

- *formate.mini_hooks.squish_stubs()* – Don't crash due to accessing an out-of-range value from a list.

0.3.1

Bugs Fixed

- *formate.isort_hook()* – Preserve aliases / re-exports (e.g. import foo as foo) in stub files, as these are necessary for type checkers to understand re-exports.

0.3.0

Enhancements

- Add support for reading the configuration from a `[tool.formate]` table in `pyproject.toml`.

0.2.5

Bugs Fixed

- *formate.mini_hooks.squish_stubs()* – Improve handling of stubs with multiple decorators and keyword-only arguments.

0.2.4

Bugs Fixed

- *formate.isort_hook()* – Correctly handle isort options which may be either a single value or a sequence of values.

0.2.3

Bugs Fixed

- *formate.mini_hooks.squish_stubs()* – Correctly handle comments and docstrings at the very top of stub files.

0.2.0

Additions

- *@formate.config.wants_filename*
- *formate.mini_hooks.squish_stubs()*

0.1.0

Initial release.

Chapter

EIGHT

Downloading source code

The `formate` source code is available on GitHub, and can be accessed from the following URL: https://github.com/python-formate/formate

If you have `git` installed, you can clone the repository with the following command:

```
$ git clone https://github.com/python-formate/formate
```

```
Cloning into 'formate'...
remote: Enumerating objects: 47, done.
remote: Counting objects: 100% (47/47), done.
remote: Compressing objects: 100% (41/41), done.
remote: Total 173 (delta 16), reused 17 (delta 6), pack-reused 126
Receiving objects: 100% (173/173), 126.56 KiB | 678.00 KiB/s, done.
Resolving deltas: 100% (66/66), done.
```

Alternatively, the code can be downloaded in a 'zip' file by clicking:
Clone or download –> Download Zip

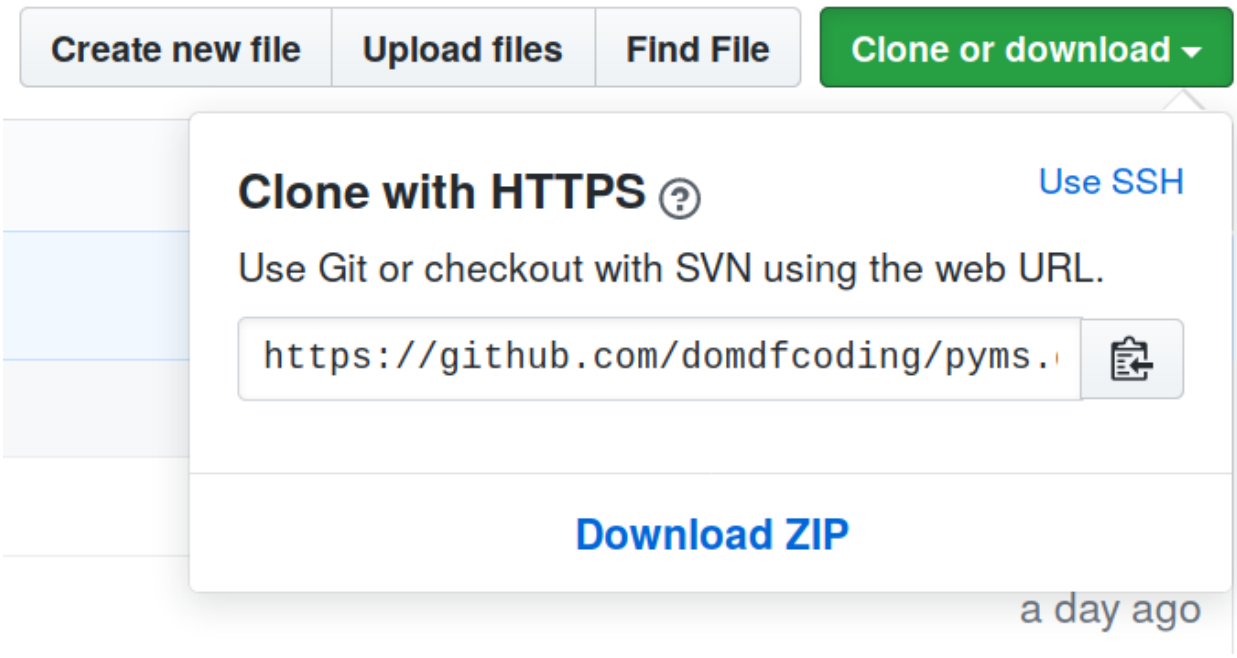

Fig. 1: Downloading a 'zip' file of the source code

8.1 Building from source

The recommended way to build `formate` is to use tox:

```
$ tox -e build
```

The source and wheel distributions will be in the directory `dist`.

If you wish, you may also use pep517.build or another **PEP 517**-compatible build tool.

Chapter

NINE

License

`formate` is licensed under the MIT License

A short and simple permissive license with conditions only requiring preservation of copyright and license notices. Licensed works, modifications, and larger works may be distributed under different terms and without source code.

Permissions

- Commercial use – The licensed material and derivatives may be used for commercial purposes.
- Modification – The licensed material may be modified.
- Distribution – The licensed material may be distributed.
- Private use – The licensed material may be used and modified in private.

Conditions

- License and copyright notice – A copy of the license and copyright notice must be included with the licensed material.

Limitations

- Liability – This license includes a limitation of liability.
- Warranty – This license explicitly states that it does NOT provide any warranty.

See more information on choosealicense.com ⇒

Part II

API Reference

Chapter

TEN

formate

Python formatting mate.

Functions:

call_hooks(hooks, source, filename)	Given a list of hooks (in order), call them in turn to reformat the source.
reformat_file(filename, config[, colour])	Reformat the given file, and show the diff if changes were made.
isort_hook(source, formate_filename[, ...])	Call isort, using the given keyword arguments as its configuration.
yapf_hook(source[, formate_global_config])	Call yapf, using the given keyword arguments as its configuration.

Classes:

Reformatter(filename, config)	Reformat a Python source file.

call_hooks (*hooks*, *source*, *filename*)

Given a list of hooks (in order), call them in turn to reformat the source.

Parameters

- **hooks** (Iterable[*Hook*])
- **source** (str) – The source to reformat.
- **filename** (Union[str, Path, PathLike]) – The name of the source file.

Return type str

Returns The reformatted source.

Changed in version 0.4.3: Added the `filename` argument.

reformat_file (*filename*, *config*, *colour=None*)

Reformat the given file, and show the diff if changes were made.

Parameters

- **filename** (Union[str, Path, PathLike]) – The filename to reformat.
- **config** (*FormateConfigDict*) – The `formate` configuration, parsed from a TOML file (or similar).
- **colour** (Optional[bool]) – Whether to force coloured output on (True) or off (False). Default None.

Return type int

class Reformatter(*filename*, *config*)

Bases: object

Reformat a Python source file.

Parameters

- **filename** (Union[str, Path, PathLike]) – The filename to reformat.
- **config** (*FormateConfigDict*) – The formate configuration, parsed from a TOML file (or similar).

Attributes:

file_to_format	The filename being reformatted, as a POSIX-style path.
filename	The filename being reformatted.
config	The formate configuration, parsed from a TOML file (or similar).

Methods:

run()	Run the reformatter.
get_diff()	Returns the diff between the original and reformatted file content.
to_string()	Return the reformatted file as a string.
to_file()	Write the reformatted source to the original file.

file_to_format

Type: PathPlus

The filename being reformatted, as a POSIX-style path.

filename

Type: str

The filename being reformatted.

config

Type: *FormateConfigDict*

The formate configuration, parsed from a TOML file (or similar).

run()

Run the reformatter.

Return type bool

Returns Whether the file was changed.

get_diff()

Returns the diff between the original and reformatted file content.

Return type str

to_string()
Return the reformatted file as a string.

Return type str

to_file()
Write the reformatted source to the original file.

isort_hook(*source*, *formate_filename*, *formate_global_config=None*, ***kwargs*)
Call isort, using the given keyword arguments as its configuration.

Parameters

- **source** (str) – The source to reformat.
- **formate_filename** (Union[str, Path, PathLike]) – The path to the file being reformatted.
- **formate_global_config** (Optional[Mapping]) – The global configuration dictionary. Optional. Default None.
- ****kwargs**

Return type str

Returns The reformatted source.

yapf_hook(*source*, *formate_global_config=None*, ***kwargs*)
Call yapf, using the given keyword arguments as its configuration.

Parameters

- **source** (str) – The source to reformat.
- **formate_global_config** (Optional[Mapping]) – The global configuration dictionary. Optional. Default None.
- ****kwargs**

If yapf_style is given as a keyword argument, use that style. If a filename is given as the style it is searched for in the current and parent directories, and the style taken from the configuration in that file.

Return type str

Returns The reformatted source.

Chapter

ELEVEN

formate.classes

Core classes.

Classes:

FormateConfigDict	typing.TypedDict representing the configuration mapping parsed from formate.toml or similar.
ExpandedHookDict	typing.TypedDict representing the expanded form of a hook in the mapping parsed from the config file.
EntryPoint(name, obj)	Represents an entry point for a hook.
Hook(name[, priority, args, kwargs, ...])	Represents a formate reformatting hook.

Data:

HooksMapping	Type hint for the hooks key of the formate configuration mapping.

typeddict FormateConfigDict

Bases: TypedDict

typing.TypedDict representing the configuration mapping parsed from formate.toml or similar.

Optional Keys

- **hooks** (Mapping[str, Union[int, *ExpandedHookDict*]]) – Mapping defining the hooks to run. Each value can either be an integer (the priority) or a *ExpandedHookDict*.
- **config** (Mapping[str, Any]) – Mapping defining the global configuration for formate.

typeddict ExpandedHookDict

Bases: TypedDict

typing.TypedDict representing the expanded form of a hook in the mapping parsed from the config file.

Required Keys

- **priority** (int) – The priority of the hook.

Optional Keys

- **args** (List[Any]) – The positional arguments passed to the hook function.
- **kwargs** (Dict[str, Any]) – The keyword arguments passed to the hook function.

HooksMapping

Type hint for the hooks key of the formate configuration mapping.

Alias of Mapping[str, Union[int, *ExpandedHookDict*]]

class EntryPoint(*name*, *obj*)

Bases: object

Represents an entry point for a hook.

Parameters

- **name** (str) – The name of the entry point. The name is normalized into lowercase, with underscores replaced by hyphens.
- **obj** (Callable[..., str]) – The object the entry point refers to.

Attributes:

name	The name of the entry point.
obj	The object the entry point refers to.

Methods:

__repr__()	Return a string representation of the *EntryPoint*.
from_dict(d)	Construct an instance of *EntryPoint* from a dictionary.
to_dict([convert_values])	Returns a dictionary containing the contents of the *EntryPoint* object.

name

Type: str

The name of the entry point. The name is normalized into lowercase, with underscores replaced by hyphens.

obj

Type: Callable[..., str]

The object the entry point refers to.

__repr__()

Return a string representation of the *EntryPoint*.

Return type str

classmethod from_dict(*d*)

Construct an instance of *EntryPoint* from a dictionary.

Parameters **d** (Mapping[str, Any]) – The dictionary.

to_dict(*convert_values=False*)

Returns a dictionary containing the contents of the *EntryPoint* object.

Parameters **convert_values** (bool) – Recursively convert values into dictionaries, lists etc. as appropriate. Default False.

Return type MutableMapping[str, Any]

class Hook(*name*, *priority=10*, *args=()*, *kwargs={}*, *entry_point=None*, *global_config={}*)

Bases: object

Represents a `formate` reformatting hook.

Parameters

- **name** (str) – The name of the entry point. The name is normalized into lowercase, with underscores replaced by hyphens.
- **priority** (int) – The priority of the hook. Default `10`.
- **args** – The positional arguments passed to the hook function. Default `()`.
- **kwargs** (Dict[str, Any]) – The keyword arguments passed to the hook function. Default `{}`.
- **entry_point** (Optional[*EntryPoint*]) – Default `None`.
- **global_config** (Mapping[str, Any]) – A read-only view on the global configuration mapping, for hooks to do with as they wish. Default `{}`.

Attributes:

name	The name of the hook.
priority	The priority of the hook.
args	The positional arguments passed to the hook function.
kwargs	The keyword arguments passed to the hook function.
global_config	A read-only view on the global configuration mapping, for hooks to do with as they wish.

Methods:

parse(data)	Parse the given mapping into *Hook*s.
__call__(source, filename)	Call the hook.
__repr__()	Return a string representation of the *Hook*.
from_dict(d)	Construct an instance of *Hook* from a dictionary.
to_dict([convert_values])	Returns a dictionary containing the contents of the *Hook* object.

name

Type: str

The name of the hook. The name is normalized into lowercase, with underscores replaced by hyphens.

priority

Type: int

The priority of the hook.

args

Type: Sequence[Any]

The positional arguments passed to the hook function.

kwargs

Type: Dict[str, Any]

The keyword arguments passed to the hook function.

global_config
Type: Mapping[str, Any]

A read-only view on the global configuration mapping, for hooks to do with as they wish.

classmethod parse(*data*)
Parse the given mapping into *Hook*s.

Parameters **data** (Mapping[str, Union[int, *ExpandedHookDict*]])

Return type Iterator[*Hook*]

__call__(*source*, *filename*)
Call the hook.

Parameters

- **source** (str) – The source to reformat.
- **filename** (Union[str, Path, PathLike]) – The name of the source file.

Return type str

Returns The reformatted source.

Raises TypeError if `entry_point` has not been set.

Changed in version 0.2.0: Added the `filename` argument.

__repr__()
Return a string representation of the *Hook*.

Return type str

classmethod from_dict(*d*)
Construct an instance of *Hook* from a dictionary.

Parameters **d** (Mapping[str, Any]) – The dictionary.

to_dict(*convert_values=False*)
Returns a dictionary containing the contents of the *Hook* object.

Parameters **convert_values** (bool) – Recursively convert values into dictionaries, lists etc. as appropriate. Default False.

Return type MutableMapping[str, Any]

Chapter

TWELVE

formate.config

Read and parse formate configuration.

Data:

_C_str	Invariant TypeVar bound to typing.Callable[..., str].

Functions:

load_toml(filename)	Load the formate configuration mapping from the given TOML file.
parse_global_config(config)	Returns a read-only view on the global configuration mapping, for hooks to do with as they wish.
parse_hooks(config)	Given a mapping parsed from a TOML file (or similar), return a list of hooks selected by the user.
wants_filename(func)	Decorator to indicate to formate that the filename being reformatted should be passed to this hook.
wants_global_config(func)	Decorator to indicate to formate that the global configuration should be passed to this hook.

_C_str = TypeVar(_C_str, bound=typing.Callable[..., str])

Type: TypeVar

Invariant TypeVar bound to typing.Callable[..., str].

load_toml(*filename*)

Load the formate configuration mapping from the given TOML file.

Parameters **filename** (Union[str, Path, PathLike])

Return type *FormateConfigDict*

parse_global_config(*config*)

Returns a read-only view on the global configuration mapping, for hooks to do with as they wish.

Parameters **config** (Mapping) – The formate configuration, parsed from a TOML file (or similar).

Return type MappingProxyType

parse_hooks(*config*)

Given a mapping parsed from a TOML file (or similar), return a list of hooks selected by the user.

Parameters **config** (Mapping) – The formate configuration, parsed from a TOML file (or similar).

Return type List[*Hook*]

wants_filename (*func*)
Decorator to indicate to `formate` that the filename being reformatted should be passed to this hook.

The configuration will be provided as the `formate_filename`: `PathLike` keyword argument.

New in version 0.2.0.

Parameters **func** (*~_C_str*)

Return type *~_C_str*

wants_global_config (*func*)
Decorator to indicate to `formate` that the global configuration should be passed to this hook.

The configuration will be provided as the `formate_global_config`: `Mapping` keyword argument.

Parameters **func** (*~_C_str*)

Return type *~_C_str*

Chapter

THIRTEEN

formate.dynamic_quotes

Applies “dynamic quotes” to Python source code.

The rules are:

- Use double quotes `"` where possible.
- Use single quotes `'` for empty strings and single characters (`a`, `\n` etc.).
- Leave the quotes unchanged for multiline strings, f strings and raw strings.

dynamic_quotes(*source*)

Reformats quotes in the given source, and returns the reformatted source.

Parameters **source** (`str`) – The source to reformat.

Return type `str`

Returns The reformatted source.

Chapter

FOURTEEN

formate.ellipses

Move ellipses (`...`) for type stubs onto the end of the stub definition.

Before:

```
def foo(value: str) -> int:
    ...
```

After:

```
def foo(value: str) -> int: ...
```

class EllipsisRewriter(*source*)

Bases: *Rewriter*

Move ellipses (`...`) for type stubs onto the end of the stub definition.

Parameters **source** (`str`) – The source to reformat.

rewrite_ellipsis(*node*)

Responsible for the actual rewriting.

ellipsis_reformat(*source*)

Move ellipses (`...`) for type stubs onto the end of the stub definition.

Before:

```
def foo(value: str) -> int:
    ...
```

After:

```
def foo(value: str) -> int: ...
```

Parameters **source** (`str`) – The source to reformat.

Return type `str`

Returns The reformatted source.

Chapter

FIFTEEN

formate.exceptions

Exception classes.

exception HookNotFoundError(*hook*)

Bases: ValueError

Exception to indicate the specified hook could not be found.

hook

Type: *Hook*

The missing hook.

Chapter

SIXTEEN

formate.imports

Converts import statements.

Classes:

CollectionsABCRewriter(source)	Identify deprecated `from collections import <abc>` imports, and rewrite them as `from collections.abc import <abc>`.

Functions:

rewrite_collections_abc_imports(source)	Identify deprecated `from collections import <abc>` imports, and rewrite them as `from collections.abc import <abc>`.

class CollectionsABCRewriter(*source*)

Bases: *Rewriter*

Identify deprecated `from collections import <abc>` imports, and rewrite them as `from collections.abc import <abc>`.

Parameters **source** (`str`) – The source to reformat.

rewrite_collections_abc_imports(*source*)

Identify deprecated `from collections import <abc>` imports, and rewrite them as `from collections.abc import <abc>`.

Parameters **source** (`str`) – The source to reformat.

Return type `str`

Returns The reformatted source.

Chapter

SEVENTEEN

formate.mini_hooks

Small but mighty hooks.

Functions:

check_ast(source)	Check the source can be parsed as a Python Abstract Syntax Tree.
noqa_reformat(source)	Pull `# noqa: ...` comments that immediately follow docstrings back up to the end of the correct line.
squish_stubs(source, formate_filename)	Squash type stubs by removing unnecessary blank lines.

check_ast(*source*)

Check the source can be parsed as a Python Abstract Syntax Tree.

Parameters **source** (str) – The source to check.

Raises **SyntaxError** – If the source is not valid Python.

Return type str

Returns The source unchanged.

noqa_reformat(*source*)

Pull `# noqa: ...` comments that immediately follow docstrings back up to the end of the correct line.

Parameters **source** (str) – The source to reformat.

Return type str

Returns The reformatted source.

squish_stubs(*source*, *formate_filename*)

Squash type stubs by removing unnecessary blank lines.

New in version 0.2.0.

Parameters

- **source** (str) – The source to check.
- **formate_filename** (Union[str, Path, PathLike]) – The name of the source file, to ensure this hook only runs on type stubs.

Return type str

Returns The reformatted source.

Chapter

EIGHTEEN

formate.reformat_generics

Formats generics (`List[...]`, `Union[...]` etc.) at the module and class level.

Example output, with a line length of `100`:

```
ParamsMappingValueType = Union[str, bytes, int, float, Iterable[Union[str, bytes, int,
↪ float]]]
Data = Union[None, str, bytes, MutableMapping[str, Any], Iterable[Tuple[str,␣
↪Optional[str]]], IO]
ParamsType = Union[
    Mapping[Union[str, bytes, int, float], ParamsMappingValueType],
    Union[str, bytes],
    Tuple[Union[str, bytes, int, float], ParamsMappingValueType],
    None
    ]
```

Functions:

reformat_generics(source[, ...])	Reformats generics (`typing.Generic`, `typing.Union`, `typing.Callable` etc.) in the given source, and returns the reformatted source.

Classes:

Generic(name, elements)	Represents a `typing.Generic`, `typing.Union`, `typing.Callable` etc.
List(elements)	Represents a list of elements, most often used within a `typing.Callable`.

reformat_generics(*source*, *formate_global_config=None*, ***kwargs*)

Reformats generics (`typing.Generic`, `typing.Union`, `typing.Callable` etc.) in the given source, and returns the reformatted source.

Parameters

- **source** (`str`) – The source to reformat.
- **formate_global_config** (`Optional`[`Mapping`]) – The global configuration dictionary. Optional. Default `None`.
- ****kwargs**

Return type `str`

Returns The reformatted source.

class Generic(*name*, *elements*)

Bases: object

Represents a typing.Generic, typing.Union, typing.Callable etc.

Parameters

- **name** (str) – The name of the Generic
- **elements** (Sequence[Union[str, *Generic*, *List*]]) – The __class_getitem__ elements of the Generic.

Methods:

format([line_offset])	Formats the *Generic*.

format(*line_offset=0*)

Formats the *Generic*.

Parameters **line_offset** (int) – Default 0.

Return type str

class List(*elements*)

Bases: object

Represents a list of elements, most often used within a typing.Callable.

Parameters **elements** (Sequence[Union[str, *Generic*, *List*]])

Chapter

NINETEEN

formate.utils

Utility functions.

Functions:

import_entry_points(hooks)	Given a list of hooks, import the corresponding entry point and return a mapping of entry point names to *EntryPoint* objects.
normalize(name)	Normalize the given name into lowercase, with underscores replaced by hyphens.
syntaxerror_for_file(filename)	Context manager to catch SyntaxError and set its filename to filename if the current filename is <unknown>.

Classes:

Rewriter(source)	ABC for rewriting Python source files from an AST and a token stream.
SyntaxTracebackHandler([exception])	Subclass of consolekit.tracebacks.TracebackHandler to additionally handle SyntaxError.

import_entry_points(*hooks*)

Given a list of hooks, import the corresponding entry point and return a mapping of entry point names to *EntryPoint* objects.

Parameters **hooks** (List[*Hook*])

Raises *HookNotFoundError* if no entry point can be found for a hook.

Return type Dict[str, *EntryPoint*]

normalize(*name*)

Normalize the given name into lowercase, with underscores replaced by hyphens.

Parameters **name** (str) – The hook name.

Return type str

syntaxerror_for_file(*filename*)

Context manager to catch SyntaxError and set its filename to filename if the current filename is <unknown>.

This is useful for syntax errors raised when parsing source into an AST.

Parameters **filename** (Union[str, Path, PathLike])

Return type Iterator

class Rewriter(*source*)

Bases: NodeVisitor

ABC for rewriting Python source files from an AST and a token stream.

Attributes:

source	The original source.
tokens	The tokenized source.
replacements	The parts of code to replace.

Methods:

rewrite()	Rewrite the source and return the new source.
record_replacement(text_range, new_source)	Record a region of text to be replaced.

source

Type: str

The original source.

tokens

Type: ASTTokens

The tokenized source.

replacements

Type: List[Tuple[Tuple[int, int], str]]

The parts of code to replace.

Each element comprises a tuple of `(start char, end char)` in *source*, and the new text to insert between these positions.

rewrite()

Rewrite the source and return the new source.

Return type str

Returns The reformatted source.

record_replacement(*text_range*, *new_source*)

Record a region of text to be replaced.

Parameters

- **text_range** (Tuple[int, int]) – The region of text to be replaced.
- **new_source** (str) – The new text for that region.

class SyntaxTracebackHandler(*exception=Abort()*)

Bases: TracebackHandler

Subclass of consolekit.tracebacks.TracebackHandler to additionally handle SyntaxError.

Python Module Index

f

Index

Symbols

A

C

D

E

F