flake8-nb Documentation

Release 0.5.3

Sebastian Weigand

Jul 07, 2023

CONTENTS:

CHAPTER

ONE

FLAKE8-NB

All Contributors

`flake8` checking for jupyter notebooks.

This tool is mainly aimed towards writing tutorials/lecture material, where one might also want to show off bad practices and/or errors, while still keeping the rest of the code clean and without adding the complexity of tooling to the readers (see docs on cell tags).

Basically this is a hack on the `flake8`'s `Application` class, which adds parsing and a cell based formatter for `*.ipynb` files.

This is **NOT A PLUGIN** but a stand alone CLI tool/pre-commit hook to be used instead of the `flake8` command/hook.

1.1 Features

- flake8 CLI tests for jupyter notebooks
- Full base functionality of `flake8` and its plugins
- Input cell based error formatting (Execution count/code cell count/total cellcount)
- Report fine tuning with cell-tags (`flake8-noqa-tags` see usage)
- pre-commit hook

1.2 Examples

1.3 Default reporting

If you had a notebook with name `example_notebook.ipynb`, where the code cell which was executed as 34th cell (`In[34]`) had the following code:

```
bad_formatted_dict = {"missing":"space"}
```

running `flake8_nb` would result in the following output.

1.3.1 Execution count

```
$ flake8_nb example_notebook.ipynb
example_notebook.ipynb#In[34]:1:31: E231 missing whitespace after ':'
```

1.4 Custom reporting

If you prefer the reports to show the cell number rather then the execution count you can use the `--notebook-cell-format` option, given that the cell is the 5th `code` cell and 10th total cell (taking `raw` and `markdown` cells into account), you will get the following output.

1.4.1 Code cell count

```
$ flake8_nb --notebook-cell-format '{nb_path}:code_cell#{code_cell_count}' example_
↪notebook.ipynb
example_notebook.ipynb:code_cell#5:1:31: E231 missing whitespace after ':'
```

1.4.2 Total cell count

```
$ flake8_nb --notebook-cell-format '{nb_path}:cell#{total_cell_count}' example_notebook.
↪ipynb
example_notebook.ipynb:cell#10:1:31: E231 missing whitespace after ':'
```

1.5 Similar projects

- nbQA: Run isort, pyupgrade, mypy, pylint, flake8, mdformat, black, blacken-docs, and more on Jupyter Notebooks

1.6 Contributors

Thanks goes to these wonderful people (emoji key):

This project follows the all-contributors specification. Contributions of any kind welcome!

CHAPTER

TWO

INSTALLATION

2.1 Stable release

To install flake8-nb, run this command in your terminal:

```
$ pip install flake8-nb
```

This is the preferred method to install flake8-nb, as it will always install the most recent stable release.

If you don't have pip installed, this Python installation guide can guide you through the process.

2.2 From sources

You can either pip install it directly from github:

```
$ pip install git+git://github.com/s-weigand/flake8-nb@<branch_name>
```

Or get the sources for flake8-nb, which can be downloaded from the Github repo.

By cloning the public repository:

```
$ git clone git://github.com/s-weigand/flake8-nb
```

Or downloading the tarball:

```
$ curl -OJL https://github.com/s-weigand/flake8-nb/tarball/main
```

Once you have a copy of the source, you can install it with:

```
$ python setup.py install
```

CHAPTER

THREE

USAGE

Since `flake8_nb` is basically a hacked version of `flake8` its usage is identically. The only key difference is the appended `_nb` is the commands and configurations name.

3.1 Command line usage

The basic usage is to call `flake8_nb` with the files/paths, which should be checked as arguments (see flake8 invocation).

```
$ flake8_nb path-to-notebooks-or-folder
```

To customize the behavior you can use the many options provided by `flake8`'s CLI. To see all the provided option just call:

```
$ flake8_nb --help
```

Additional flags/options provided by `flake8_nb`:

- **`--keep-parsed-notebooks`**
  If this flag is activated the the parsed notebooks will be kept and the path they were saved in will be displayed, for further debugging or trouble shooting.
- **`--notebook-cell-format`**
  Template string used to format the filename and cell part of error report. Possible variables which will be replaced are `nb_path`, `exec_count`, `code_cell_count` and `total_cell_count`.

3.2 Project wide configuration

Configuration of a project can be saved in one of the following files `setup.cfg`, `tox.ini` or `.flake8_nb`, on the top level of your project (see flake8 configuration).

```
[flake8_nb]
; Default values
keep_parsed_notebooks = False
notebook_cell_format = {nb_path}#In[{exec_count}]
```

For a detailed explanation on how to use and configure it, you can consult the official flake8 documentation

3.3 Per cell/line configuration

There are multiple ways to fine grade configure `flake8_nb` on a line or cell basis.

3.3.1 flake8 `noqa` comments

The most intuitive way for experienced `flake8` users is to utilize the known flake8 noqa comment on a line, to ignore specific or all errors, `flake8` would report on that given line.

Note: If a normal `flake8 noqa comment` ends with a string, which doesn't match the error code pattern (`\w+\d+`), this comment will be ignored.

3.3.2 Cell tags

Cell tags are meta information, which can be added to cells, to augment their behavior (for `jupyterlab<2.0`, you will need to install jupyterlab-celltags). Depending on the editor you use for the notebook, they aren't directly visible, which is a nice way to hide certain internals which aren't important for the user/reader. For example if write a book like notebook and want to demonstrate some bad code examples an still pass your `flake8_nb` tests you can use `flake8-noqa-tags`. Or if you want to demonstrate a raised exception and still want then whole notebook to be executed when you run all cells, you can use the `raises-exception` tag.

The patterns for `flake8-noqa-tags` are the following:

- **`flake8-noqa-cell`**
  ignores all reports from a cell
- **`flake8-noqa-cell-<rule1>-<rule2>`**
  ignores given rules for the cell i.e. `flake8-noqa-cell-F401-F811`
- **`flake8-noqa-line-<line_nr>`**
  ignores all reports from a given line in a cell, i.e. `flake8-noqa-line-1`
- **`flake8-noqa-line-<line_nr>-<rule1>-<rule2>`**
  ignores given rules from a given line for the cell i.e. `flake8-noqa-line-1-F401-F811`

3.3.3 Inline cell tags

If you want your users/reader to directly see which `flake8` rules are ignored, you can also use the `flake8-noqa-tag` pattern as comment in a cell.

Note: If you use jupyter magic to run code other than Python (i.e. `%%bash`) you should ignore the whole cell with `flake8-noqa-cell`.

3.4 As pre-commit hook

Add the following to your `.pre-commit-config.yaml` file:

```
- repo: https://github.com/s-weigand/flake8-nb
  rev: 0.5.3 # specify version here
  hooks:
  - id: flake8-nb
```

See pre-commit docs for more on pre-commit.

CHAPTER

FOUR

EXAMPLES

4.1 This notebook demonstrates `flake8_nb` reporting

The next cell should report `F401 'not_a_package' imported but unused`

[1]:
```
import not_a_package
```

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
<ipython-input-1-d9fbe3554078> in <module>
----> 1 import not_a_package

ModuleNotFoundError: No module named 'not_a_package'
```

The next cell should report `E231 missing whitespace after ':'`

[2]:
```
{"1":1}
```

[2]:
```
{'1': 1}
```

The next cell should not be reported, since it is valid syntax

[3]:
```
def func():
    return "foo"
```

The next cell should not be reported, since it is valid syntax

[4]:
```
class Bar:
    def foo(self):
        return "foo"
```

The next cell should be ignored in the generated intermediate `*.py` file since it is empty

[ ]:
```

```

The next cell should report `E231 missing whitespace after ':'`

[5]:
```
{"1":1}
```

[5]:
```
{'1': 1}
```

4.1.1 Report using execution count

[6]:
```
!flake8_nb notebook_with_out_flake8_tags.ipynb
```

```
notebook_with_out_flake8_tags.ipynb#In[1]:1:1: F401 'not_a_package' imported but unused
notebook_with_out_flake8_tags.ipynb#In[2]:1:5: E231 missing whitespace after ':'
notebook_with_out_flake8_tags.ipynb#In[5]:1:5: E231 missing whitespace after ':'
```

4.1.2 Report using code cell count

[7]:
```
!flake8_nb --notebook-cell-format '{nb_path}:code_cell#{code_cell_count}' notebook_with_
↪out_flake8_tags.ipynb
```

```
'notebook_with_out_flake8_tags.ipynb:code_cell#1':1:1: F401 'not_a_package' imported but␣
↪unused
'notebook_with_out_flake8_tags.ipynb:code_cell#2':1:5: E231 missing whitespace after ':'
'notebook_with_out_flake8_tags.ipynb:code_cell#6':1:5: E231 missing whitespace after ':'
```

4.1.3 Report using total cell count

[8]:
```
!flake8_nb --notebook-cell-format '{nb_path}:cell#{total_cell_count}' notebook_with_out_
↪flake8_tags.ipynb
```

```
'notebook_with_out_flake8_tags.ipynb:cell#3':1:1: F401 'not_a_package' imported but␣
↪unused
'notebook_with_out_flake8_tags.ipynb:cell#5':1:5: E231 missing whitespace after ':'
'notebook_with_out_flake8_tags.ipynb:cell#14':1:5: E231 missing whitespace after ':'
```

4.2 This notebook demonstrates `flake8_nb` reporting with `flake8-tags`

4.2.1 Testing Celltags

The next cell should not report `F401 'not_a_package' imported but unused`, due to the cell tag `flake8-noqa-cell-F401`.

But it will report `E231 missing whitespace after ':'`, for line 2.

[1]:
```
import not_a_package
{"1":1}
```

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
<ipython-input-1-e5f2776a04db> in <module>
----> 1 import not_a_package
      2 {"1":1}

ModuleNotFoundError: No module named 'not_a_package'
```

The next cell should not report `E231 missing whitespace after ':'` for line 1, due to the line tag `flake8-noqa-line-1-E231`.

But it will report `E231 missing whitespace after ':'`, for line 2.

[2]:
```
{"2":1}
{"2":2}
```

[2]:
```
{'2': 2}
```

The next cell should not report `E231 missing whitespace after ':'`, due to the line tag `flake8-noqa-cell-E231`.

[3]:
```
{"3":1}
{"3":2}
```

[3]:
```
{'3': 2}
```

4.2.2 Testing inline Celltags

The next cell should not report `F401 'not_a_package' imported but unused`, `F811 redefinition of unused 'not_a_package'` and `E402 module level import not at top of file`, due to the inline cell tag `flake8-noqa-cell-E402-F401-F811`.

But it will report `E231 missing whitespace after ':'`, for line 3.

[4]:
```
# flake8-noqa-cell-E402-F401-F811
import not_a_package
{"4":1}
```

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
<ipython-input-4-4da0e31bdfa0> in <module>
      1 # flake8-noqa-cell-E402-F401-F811
----> 2 import not_a_package
      3 {"4":1}

ModuleNotFoundError: No module named 'not_a_package'
```

The next cell should not report `E231 missing whitespace after ':'` for line 2, due to the inline line tag `flake8-noqa-line-2-E231`.

But it will report `E231 missing whitespace after ':'`, for line 3.

[5]:
```
# flake8-noqa-line-2-E231
{"5":1}
{"5":2}
```

[5]:
```
{'5': 2}
```

The next cell should not report `E231 missing whitespace after ':'`, due to the inline cell tag `flake8-noqa-cell-E231`.

[6]:
```
# flake8-noqa-cell-E231
{"6":1}
{"6":2}
```

[6]:
```
{'6': 2}
```

4.2.3 Testing normal flake8 noqa comments

The next cell should not report `E231 missing whitespace after ':'` for line 1, due to the flake8 noqa comment `#noqa: E231`.

But it will report `E231 missing whitespace after ':'`, for line 2.

[7]:
```
{"5":1}  # noqa: E231
{"5":2}
```

[7]:
```
{'5': 2}
```

4.2.4 Report using execution count

[8]:
```
!flake8_nb notebook_with_flake8_tags.ipynb
```

```
notebook_with_flake8_tags.ipynb#In[1]:2:5: E231 missing whitespace after ':'
notebook_with_flake8_tags.ipynb#In[2]:2:5: E231 missing whitespace after ':'
notebook_with_flake8_tags.ipynb#In[4]:3:5: E231 missing whitespace after ':'
notebook_with_flake8_tags.ipynb#In[5]:3:5: E231 missing whitespace after ':'
notebook_with_flake8_tags.ipynb#In[7]:2:5: E231 missing whitespace after ':'
```

4.2.5 Report using code cell count

[9]:
```
!flake8_nb --notebook-cell-format '{nb_path}:code_cell#{code_cell_count}' notebook_with_
↪flake8_tags.ipynb
```

```
'notebook_with_flake8_tags.ipynb:code_cell#1':2:5: E231 missing whitespace after ':'
'notebook_with_flake8_tags.ipynb:code_cell#2':2:5: E231 missing whitespace after ':'
'notebook_with_flake8_tags.ipynb:code_cell#4':3:5: E231 missing whitespace after ':'
'notebook_with_flake8_tags.ipynb:code_cell#5':3:5: E231 missing whitespace after ':'
'notebook_with_flake8_tags.ipynb:code_cell#7':2:5: E231 missing whitespace after ':'
```

4.2.6 Report using total cell count

[10]:
```
!flake8_nb --notebook-cell-format '{nb_path}:cell#{total_cell_count}' notebook_with_
↪flake8_tags.ipynb
```

```
'notebook_with_flake8_tags.ipynb:cell#4':2:5: E231 missing whitespace after ':'
'notebook_with_flake8_tags.ipynb:cell#6':2:5: E231 missing whitespace after ':'
'notebook_with_flake8_tags.ipynb:cell#11':3:5: E231 missing whitespace after ':'
'notebook_with_flake8_tags.ipynb:cell#13':3:5: E231 missing whitespace after ':'
'notebook_with_flake8_tags.ipynb:cell#18':2:5: E231 missing whitespace after ':'
```

CHAPTER

FIVE

INNER WORKINGS

This is the detailed documentation of the inner workings of `flake8_nb`.

flake8_nb	Top-level package for flake8-nb.

5.1 flake8_nb

Top-level package for flake8-nb.

Modules

flake8_nb.flake8_integration	Package containing code to integrate the parserers and hacking flake8.
flake8_nb.parsers	Package responsible for transforming notebooks to valid python files.

5.1.1 flake8_integration

Package containing code to integrate the parserers and hacking flake8.

Modules

flake8_nb.flake8_integration.cli	Module containing the notebook gatherer and hack of flake8.
flake8_nb.flake8_integration.formatter	Module containing the report formatter.

cli

Module containing the notebook gatherer and hack of flake8.

This is the main implementation of `flake8_nb`, it relies on overwriting `flake8` 's CLI default options, searching and parsing `*.ipynb` files and injecting the parsed files, during the loading of the CLI argv and config of `flake8`.

Functions

Summary

get_notebooks_from_args	Extract the absolute paths to notebooks.
hack_config_module	Create hacked version of `flake8.options.config` at runtime.
hack_option_manager_generate_versions	Closure to prepend the flake8 version to option_manager.generate_versions .

get_notebooks_from_args

get_notebooks_from_args(*args: list[str]*, *exclude: list[str] = ['*.tox/*', '*.ipynb_checkpoints*']*) → tuple[list[str], list[str]]

Extract the absolute paths to notebooks.

The paths are relative to the current directory or to the CLI passes files/folder and returned as list.

Parameters

- **args** (*list[str]*) – The left over arguments that were not parsed by `option_manager`
- **exclude** (*list[str]*) – File-/Folderpatterns that should be excluded, by default [".*tox/*", ".*ipynb_checkpoints*"]

Returns
List of found notebooks absolute paths.

Return type
tuple[list[str], list[str]]

hack_config_module

hack_config_module() → None

Create hacked version of `flake8.options.config` at runtime.

Since flake8>=5.0.0 uses hardcoded `"flake8"` to discover the config we replace with it with `"flake8_nb"` to create our own hacked version and replace the references to the original module with the hacked one.

See:
https://github.com/s-weigand/flake8-nb/issues/249 https://github.com/s-weigand/flake8-nb/issues/254

hack_option_manager_generate_versions

hack_option_manager_generate_versions(*generate_versions: Callable[[...], str]*) → Callable[[...], str]

Closure to prepend the flake8 version to option_manager.generate_versions .

Parameters
generate_versions (*Callable[..., str]*) – option_manager.generate_versions of flake8.options.manager.OptionManager

Returns
hacked_generate_versions

Return type
Callable[..., str]

Classes

Summary

Flake8NbApplication	Subclass of `flake8.main.application.Application`.

Flake8NbApplication

class Flake8NbApplication(*program: str = 'flake8_nb'*, *version: str = '0.5.3'*)

Subclass of `flake8.main.application.Application`.

It overwrites the default options and an injection of intermediate parsed `*.ipynb` files to be checked.

Hacked initialization of flake8.Application.

Parameters

- **program** (*str*) – Application name, by default "flake8_nb"
- **version** (*str*) – Application version, by default __version__

Attributes Summary

`formatter`	Generic output formatting.

Methods Summary

apply_hacks	Apply hacks to flake8 adding options and changing the application name + version.
exit	Handle finalization and exiting the program.
exit_code	Return the program exit code.
find_plugins	Find and load the plugins for this application.
hack_args	Update args with `*.ipynb` files.
hack_flake8_program_and_version	Hack to overwrite the program name and version of flake8.
hack_options	Overwrite `flake8`'s default options, with `flake8_nb` defaults.
hacked_register_plugin_options	Register options provided by plugins to our option manager.
initialize	Initialize the application to be run.
make_file_checker_manager	Initialize our FileChecker Manager.
make_formatter	Initialize a formatter based on the parsed options.
make_guide	Initialize our StyleGuide.
parse_configuration_and_cli	Parse configuration files and the CLI options.
parse_configuration_and_cli_legacy	Parse configuration files and the CLI options.
parse_preliminary_options	Get preliminary options from the CLI, pre-plugin-loading.
register_plugin_options	Register options provided by plugins to our option manager.
report	Report errors, statistics, and benchmarks.
report_benchmarks	Aggregate, calculate, and report benchmarks for this run.
report_errors	Report all the errors found by flake8 3.0.
report_statistics	Aggregate and report statistics from this run.
run	Run our application.
run_checks	Run the actual checks with the FileChecker Manager.
set_flake8_option	Overwrite flake8 options.

apply_hacks

`Flake8NbApplication.`**`apply_hacks`**`()` → None

Apply hacks to flake8 adding options and changing the application name + version.

exit

`Flake8NbApplication.`**`exit`**`()` → None

Handle finalization and exiting the program.

This should be the last thing called on the application instance. It will check certain options and exit appropriately.

Raises

`SystemExit` – For flake8>=5.0.0

exit_code

Flake8NbApplication.**exit_code**() → int

Return the program exit code.

find_plugins

Flake8NbApplication.**find_plugins**(*cfg: RawConfigParser*, *cfg_dir: str*, ***, *enable_extensions: str | None*, *require_plugins: str | None*) → None

Find and load the plugins for this application.

Set `plugins` based on loaded plugins.

hack_args

static Flake8NbApplication.**hack_args**(*args: list[str]*, *exclude: list[str]*) → list[str]

Update args with `*.ipynb` files.

Checks the passed args if `*.ipynb` can be found and appends intermediate parsed files to the list of files, which should be checked.

Parameters

- **args** (*list[str]*) – List of commandline arguments provided to `flake8_nb`
- **exclude** (*list[str]*) – File-/Folderpatterns that should be excluded

Returns

The original args + intermediate parsed `*.ipynb` files.

Return type

list[str]

hack_flake8_program_and_version

Flake8NbApplication.**hack_flake8_program_and_version**(*program: str*, *version: str*) → None

Hack to overwrite the program name and version of flake8.

This is needed because those values are hard coded at creation of *self.option_manager*.

Parameters

- **program** (*str*) – Name of the program
- **version** (*str*) – Version of the program

hack_options

Flake8NbApplication.**hack_options**() → None

Overwrite `flake8`'s default options, with `flake8_nb` defaults.

hacked_register_plugin_options

Flake8NbApplication.**hacked_register_plugin_options**() → None

Register options provided by plugins to our option manager.

initialize

Flake8NbApplication.**initialize**(*argv: Sequence[str]*) → None

Initialize the application to be run.

This finds the plugins, registers their options, and parses the command-line arguments.

make_file_checker_manager

Flake8NbApplication.**make_file_checker_manager**() → None

Initialize our FileChecker Manager.

make_formatter

Flake8NbApplication.**make_formatter**() → None

Initialize a formatter based on the parsed options.

make_guide

Flake8NbApplication.**make_guide**() → None

Initialize our StyleGuide.

parse_configuration_and_cli

Flake8NbApplication.**parse_configuration_and_cli**(*cfg: configparser.RawConfigParser*, *cfg_dir: str*, *argv: list[str]*) → None

Parse configuration files and the CLI options.

Parameters

- **cfg** (*configparser.RawConfigParser*) – Config parser instance
- **cfg_dir** (*str*) – Dir the the config is in.
- **argv** (*list[str]*) – CLI args

Raises

SystemExit – If `--bug-report` option is passed to the CLI.

parse_configuration_and_cli_legacy

Flake8NbApplication.**parse_configuration_and_cli_legacy**(*config_finder: config.ConfigFileFinder*, *argv: list[str]*) → None

Parse configuration files and the CLI options.

Parameters

- **config_finder** (*config.ConfigFileFinder*) – The finder for finding and reading configuration files.
- **argv** (*list[str]*) – Command-line arguments passed in directly.

parse_preliminary_options

Flake8NbApplication.**parse_preliminary_options**(*argv: Sequence[str]*) → Tuple[Namespace, List[str]]

Get preliminary options from the CLI, pre-plugin-loading.

We need to know the values of a few standard options so that we can locate configuration files and configure logging.

Since plugins aren't loaded yet, there may be some as-yet-unknown options; we ignore those for now, they'll be parsed later when we do real option parsing.

Parameters

argv – Command-line arguments passed in directly.

Returns

Populated namespace and list of remaining argument strings.

register_plugin_options

Flake8NbApplication.**register_plugin_options**() → None

Register options provided by plugins to our option manager.

report

Flake8NbApplication.**report**() → None

Report errors, statistics, and benchmarks.

report_benchmarks

Flake8NbApplication.**report_benchmarks**() → None

Aggregate, calculate, and report benchmarks for this run.

report_errors

Flake8NbApplication.**report_errors**() → None

Report all the errors found by flake8 3.0.

This also updates the `result_count` attribute with the total number of errors, warnings, and other messages found.

report_statistics

Flake8NbApplication.**report_statistics**() → None

Aggregate and report statistics from this run.

run

Flake8NbApplication.**run**(*argv: Sequence[str]*) → None

Run our application.

This method will also handle KeyboardInterrupt exceptions for the entirety of the flake8 application. If it sees a KeyboardInterrupt it will forcibly clean up the `Manager`.

run_checks

Flake8NbApplication.**run_checks**() → None

Run the actual checks with the FileChecker Manager.

This method encapsulates the logic to make a `Manger` instance run the checks it is managing.

set_flake8_option

Flake8NbApplication.**set_flake8_option**(*long_option_name: str*, **args: Any*, ***kwargs: Any*) → None

Overwrite flake8 options.

First deletes and than reads an option to *flake8*'s cli options, if it was present. If the option wasn't present, it just adds it.

Parameters

- **long_option_name** (*str*) – Long name of the flake8 cli option.
- **args** (*Tuple[Any]*) – Arbitrary args
- **kwargs** (*Dict[str, Any]*) – Arbitrary kwargs

Methods Documentation

apply_hacks() → None

Apply hacks to flake8 adding options and changing the application name + version.

exit() → None

Handle finalization and exiting the program.

This should be the last thing called on the application instance. It will check certain options and exit appropriately.

Raises

SystemExit – For flake8>=5.0.0

exit_code() → int

Return the program exit code.

find_plugins(*cfg: RawConfigParser*, *cfg_dir: str*, *, *enable_extensions: str | None*, *require_plugins: str | None*) → None

Find and load the plugins for this application.

Set `plugins` based on loaded plugins.

static hack_args(*args: list[str]*, *exclude: list[str]*) → list[str]

Update args with `*.ipynb` files.

Checks the passed args if `*.ipynb` can be found and appends intermediate parsed files to the list of files, which should be checked.

Parameters

- **args** (*list[str]*) – List of commandline arguments provided to `flake8_nb`
- **exclude** (*list[str]*) – File-/Folderpatterns that should be excluded

Returns

The original args + intermediate parsed `*.ipynb` files.

Return type

list[str]

hack_flake8_program_and_version(*program: str*, *version: str*) → None

Hack to overwrite the program name and version of flake8.

This is needed because those values are hard coded at creation of *self.option_manager*.

Parameters

- **program** (*str*) – Name of the program
- **version** (*str*) – Version of the program

hack_options() → None

Overwrite `flake8`'s default options, with `flake8_nb` defaults.

hacked_register_plugin_options() → None

Register options provided by plugins to our option manager.

initialize(*argv: Sequence[str]*) → None

Initialize the application to be run.

This finds the plugins, registers their options, and parses the command-line arguments.

make_file_checker_manager() → None

Initialize our FileChecker Manager.

make_formatter() → None

Initialize a formatter based on the parsed options.

make_guide() → None

Initialize our StyleGuide.

parse_configuration_and_cli(*cfg: configparser.RawConfigParser*, *cfg_dir: str*, *argv: list[str]*) → None

Parse configuration files and the CLI options.

Parameters

- **cfg** (*configparser.RawConfigParser*) – Config parser instance
- **cfg_dir** (*str*) – Dir the the config is in.
- **argv** (*list[str]*) – CLI args

Raises

SystemExit – If `--bug-report` option is passed to the CLI.

parse_configuration_and_cli_legacy(*config_finder: config.ConfigFileFinder*, *argv: list[str]*) → None

Parse configuration files and the CLI options.

Parameters

- **config_finder** (*config.ConfigFileFinder*) – The finder for finding and reading configuration files.
- **argv** (*list[str]*) – Command-line arguments passed in directly.

parse_preliminary_options(*argv: Sequence[str]*) → Tuple[Namespace, List[str]]

Get preliminary options from the CLI, pre-plugin-loading.

We need to know the values of a few standard options so that we can locate configuration files and configure logging.

Since plugins aren't loaded yet, there may be some as-yet-unknown options; we ignore those for now, they'll be parsed later when we do real option parsing.

Parameters

argv – Command-line arguments passed in directly.

Returns

Populated namespace and list of remaining argument strings.

register_plugin_options() → None

Register options provided by plugins to our option manager.

report() → None

Report errors, statistics, and benchmarks.

report_benchmarks() → None

Aggregate, calculate, and report benchmarks for this run.

report_errors() → None

Report all the errors found by flake8 3.0.

This also updates the `result_count` attribute with the total number of errors, warnings, and other messages found.

report_statistics() → None

Aggregate and report statistics from this run.

run(*argv: Sequence[str]*) → None

Run our application.

This method will also handle KeyboardInterrupt exceptions for the entirety of the flake8 application. If it sees a KeyboardInterrupt it will forcibly clean up the `Manager`.

run_checks() → None

Run the actual checks with the FileChecker Manager.

This method encapsulates the logic to make a `Manger` instance run the checks it is managing.

set_flake8_option(*long_option_name: str*, **args: Any*, ***kwargs: Any*) → None

Overwrite flake8 options.

First deletes and than reads an option to *flake8*'s cli options, if it was present. If the option wasn't present, it just adds it.

Parameters

- **long_option_name** (*str*) – Long name of the flake8 cli option.
- **args** (*Tuple[Any]*) – Arbitrary args
- **kwargs** (*Dict[str, Any]*) – Arbitrary kwargs

formatter

Module containing the report formatter.

This also includes the code to map parsed error back to the original notebook and the cell the code in.

Functions

Summary

map_notebook_error	Map the violation caused in an intermediate file back to its cause.

map_notebook_error

map_notebook_error(*violation: Violation*, *format_str: str*) → tuple[str, int] | None

Map the violation caused in an intermediate file back to its cause.

The cause is resolved as the notebook, the input cell and the respective line number in that cell.

Parameters

- **violation** (*Violation*) – Reported violation from checking the parsed notebook
- **format_str** (*str*) – Format string used to format the notebook path and cell reporting.

Returns

(filename, input_cell_line_number) `filename` being the name of the original notebook and the input cell were the violation was reported. `input_cell_line_number` line number in the input cell were the violation was reported.

Return type

tuple[str, int] | None

Classes

Summary

IpynbFormatter	Default flake8_nb formatter for jupyter notebooks.

IpynbFormatter

class IpynbFormatter(*options: Namespace*)

Default flake8_nb formatter for jupyter notebooks.

If the file to be formatted is a `*.py` file, it uses flake8's default formatter.

Initialize with the options parsed from config and cli.

This also calls a hook, *after_init()*, so subclasses do not need to call super to call this method.

Parameters

options – User specified configuration parsed from both configuration files and the command-line interface.

Attributes Summary

error_format

Methods Summary

after_init	Check for a custom format string.
beginning	Notify the formatter that we're starting to process a file.
finished	Notify the formatter that we've finished processing a file.
format	Format the error detected by a flake8 checker.
handle	Handle an error reported by Flake8.
show_benchmarks	Format and print the benchmarks.
show_source	Show the physical line generating the error.
show_statistics	Format and print the statistics.
start	Prepare the formatter to receive input.
stop	Clean up after reporting is finished.
write	Write the line either to the output file or stdout.

after_init

IpynbFormatter.**after_init**() → None

Check for a custom format string.

beginning

IpynbFormatter.**beginning**(*filename: str*) → None

Notify the formatter that we're starting to process a file.

Parameters

filename – The name of the file that Flake8 is beginning to report results from.

finished

IpynbFormatter.**finished**(*filename: str*) → None

Notify the formatter that we've finished processing a file.

Parameters

filename – The name of the file that Flake8 has finished reporting results from.

format

IpynbFormatter.**format**(*violation: Violation*) → str | None

Format the error detected by a flake8 checker.

Depending on if the violation was caused by a `*.py` file or by a parsed notebook.

Parameters

violation (*Violation*) – Error a checker reported.

Returns

Formatted error message, which will be displayed in the terminal.

Return type

str | None

handle

IpynbFormatter.**handle**(*error: Violation*) → None

Handle an error reported by Flake8.

This defaults to calling *format()*, *show_source()*, and then *write()*. To extend how errors are handled, override this method.

Parameters

error – This will be an instance of `Violation`.

show_benchmarks

IpynbFormatter.**show_benchmarks**(*benchmarks: List[Tuple[str, float]]*) → None

Format and print the benchmarks.

show_source

IpynbFormatter.**show_source**(*error: Violation*) → str | None

Show the physical line generating the error.

This also adds an indicator for the particular part of the line that is reported as generating the problem.

Parameters

error – This will be an instance of `Violation`.

Returns

The formatted error string if the user wants to show the source. If the user does not want to show the source, this will return `None`.

show_statistics

IpynbFormatter.**show_statistics**(*statistics: Statistics*) → None

Format and print the statistics.

start

IpynbFormatter.**start**() → None

Prepare the formatter to receive input.

This defaults to initializing `output_fd` if `filename`

stop

IpynbFormatter.**stop**() → None

Clean up after reporting is finished.

write

IpynbFormatter.**write**(*line: str | None*, *source: str | None*) → None

Write the line either to the output file or stdout.

This handles deciding whether to write to a file or print to standard out for subclasses. Override this if you want behaviour that differs from the default.

Parameters

- **line** – The formatted string to print or write.
- **source** – The source code that has been formatted and associated with the line of output.

Methods Documentation

after_init() → None

Check for a custom format string.

beginning(*filename: str*) → None

Notify the formatter that we're starting to process a file.

Parameters

filename – The name of the file that Flake8 is beginning to report results from.

finished(*filename: str*) → None

Notify the formatter that we've finished processing a file.

Parameters

filename – The name of the file that Flake8 has finished reporting results from.

format(*violation: Violation*) → str | None

Format the error detected by a flake8 checker.

Depending on if the violation was caused by a `*.py` file or by a parsed notebook.

Parameters

violation (*Violation*) – Error a checker reported.

Returns

Formatted error message, which will be displayed in the terminal.

Return type

str | None

handle(*error: Violation*) → None

Handle an error reported by Flake8.

This defaults to calling *format()*, *show_source()*, and then *write()*. To extend how errors are handled, override this method.

Parameters

error – This will be an instance of `Violation`.

show_benchmarks(*benchmarks: List[Tuple[str, float]]*) → None

Format and print the benchmarks.

show_source(*error: Violation*) → str | None

Show the physical line generating the error.

This also adds an indicator for the particular part of the line that is reported as generating the problem.

Parameters

error – This will be an instance of `Violation`.

Returns

The formatted error string if the user wants to show the source. If the user does not want to show the source, this will return `None`.

show_statistics(*statistics: Statistics*) → None

Format and print the statistics.

start() → None

Prepare the formatter to receive input.

This defaults to initializing `output_fd` if `filename`

stop() → None

Clean up after reporting is finished.

write(*line: str | None*, *source: str | None*) → None

Write the line either to the output file or stdout.

This handles deciding whether to write to a file or print to standard out for subclasses. Override this if you want behaviour that differs from the default.

Parameters

- **line** – The formatted string to print or write.
- **source** – The source code that has been formatted and associated with the line of output.

5.1.2 parsers

Package responsible for transforming notebooks to valid python files.

Modules

flake8_nb.parsers.cell_parsers	Module containing parsers for notebook cells.
flake8_nb.parsers.notebook_parsers	Module for parsing whole jupyter notebooks.

cell_parsers

Module containing parsers for notebook cells.

This also includes parsers for the cell and inline tags. It heavily utilizes the mutability of lists.

Functions

Summary

extract_flake8_inline_tags	Extract flake8-tags which were used as comment in a cell.
extract_flake8_tags	Extract all tag that start with 'flake8-noqa-' from a cell.
extract_inline_flake8_noqa	Extract flake8 noqa rules from normal flake8 comments .
flake8_tag_to_rules_dict	Parse a flake8 tag to a `rules_dict`.
generate_rules_list	Generate a List of rules from `rules_dict`.
get_flake8_rules_dict	Parse all flake8 tags of a cell to a `rules_dict`.
notebook_cell_to_intermediate_dict	Parse `notebook_cell` to a dict.
update_inline_flake8_noqa	Update `source_line` with flake8 noqa comments.
update_rules_dict	Update the rules dict `total_rules_dict` with `new_rules_dict`.

extract_flake8_inline_tags

extract_flake8_inline_tags(*notebook_cell: NotebookCell*) → list[str]

Extract flake8-tags which were used as comment in a cell.

Parameters
notebook_cell (*NotebookCell*) – Dict representation of a notebook cell as parsed from JSON.

Returns
List of all inline tags in the given cell, which matched `FLAKE8_INLINE_TAG_PATTERN`.

Return type
list[str]

extract_flake8_tags

extract_flake8_tags(*notebook_cell: NotebookCell*) → list[str]

Extract all tag that start with 'flake8-noqa-' from a cell.

Parameters
notebook_cell (*NotebookCell*) – Dict representation of a notebook cell as parsed from JSON.

Returns
List of all tags in the given cell, which started with 'flake8-noqa-'.

Return type
list[str]

extract_inline_flake8_noqa

extract_inline_flake8_noqa(*source_line: str*) → list[str]

Extract flake8 noqa rules from normal flake8 comments .

Parameters
source_line (*str*) – Single line of sourcecode from a cell.

Returns
List of flake8 rules.

Return type
list[str]

flake8_tag_to_rules_dict

flake8_tag_to_rules_dict(*flake8_tag: str*) → Dict[str, List[str]]

Parse a flake8 tag to a `rules_dict`.

`rules_dict` contains lists of rules, depending on if the tag is a cell or a line tag.

Parameters
flake8_tag (*str*) – String of a flake8-tag.

Returns
Dict with cell and line rules. Line rules have the line number as key and cell rules have 'cell as key'.

Return type
RulesDict

See also:

get_flake8_rules_dict

generate_rules_list

generate_rules_list(*source_index: int*, *rules_dict: RulesDict*) → list[str]

Generate a List of rules from `rules_dict`.

This list should be applied to the line at `source_index`.

Parameters

- **source_index** (*int*) – Index of the source code line.
- **rules_dict** (*RulesDict*) – Dict containing lists of rules, depending on if the tag is a cell or a line tag.

Returns
List of rules which should be applied to the line at `source_index`.

Return type
list[str]

See also:

flake8_tag_to_rules_dict, *get_flake8_rules_dict*

get_flake8_rules_dict

get_flake8_rules_dict(*notebook_cell: Dict[str, Any]*) → Dict[str, List[str]]

Parse all flake8 tags of a cell to a `rules_dict`.

`rules_dict` contains lists of rules, depending on if the tag is a cell or a line tag.

Parameters
notebook_cell (*NotebookCell*) – Dict representation of a notebook cell as parsed from JSON.

Returns
Dict with all cell and line rules. Line rules have the line number as key and cell rules have 'cell as key'.

Return type
RulesDict

See also:

flake8_tag_to_rules_dict, *update_rules_dict*

notebook_cell_to_intermediate_dict

notebook_cell_to_intermediate_dict(*notebook_cell: NotebookCell*) → dict[str, *CellId* | str | int]

Parse `notebook_cell` to a dict.

That dict can later be written to a intermediate_py_file.

Parameters
notebook_cell (*NotebookCell*) – Dict representation of a notebook cell as parsed from JSON.

Returns
Dict which has the keys 'code', 'input_name' and 'code'. `code`,``input_name`` is a str of the code cells `In[\d\*]` name and `lines_of_code` is the number of lines of corresponding parsed parsed notebook cell.

Return type
dict[str, *CellId* | str | int]

See also:

update_inline_flake8_noqa, *flake8_nb.parsers.notebook_parsers.create_intermediate_py_file*

update_inline_flake8_noqa

update_inline_flake8_noqa(*source_line: str*, *rules_list: list[str]*) → str

Update `source_line` with flake8 noqa comments.

This is done extraction flake8-tags as well as inline flake8 comments.

Parameters

- **source_line** (*str*) – Single line of sourcecode from a cell.
- **rules_list** (*list[str]*) – List of rules which should be applied to `source_line`.

Returns
`source_line` with flake8 noqa comments.

Return type
str

See also:

generate_rules_list

update_rules_dict

update_rules_dict(*total_rules_dict: Dict[str, List[str]]*, *new_rules_dict: Dict[str, List[str]]*) → None

Update the rules dict `total_rules_dict` with `new_rules_dict`.

If any entry of a key is 'noqa' (ignore all), the rules will be set to be only 'noqa'.

Parameters

- **total_rules_dict** (*RulesDict*) – `rules_dict` which should be updated.

- **new_rules_dict** (*RulesDict*) – `rules_dict` which should be used to update `total_rules_dict`.

See also:

flake8_tag_to_rules_dict, *get_flake8_rules_dict*

Exceptions

Exception Summary

InvalidFlake8TagWarning	Warning thrown when a tag is badly formatted.

InvalidFlake8TagWarning

exception InvalidFlake8TagWarning(*flake8_tag: str*)

Warning thrown when a tag is badly formatted.

When a cell tag starts with 'flake8-noqa-' but doesn't match the correct pattern needed for cell tags. This is used to show users that they have a typo in their tags.

Create InvalidFlake8TagWarning.

Parameters

flake8_tag (*str*) – Used improperly formatted flake8-nb tag

notebook_parsers

Module for parsing whole jupyter notebooks.

This utilizes `flake8_nb.parser.cell_parsers`.

Functions

Summary

convert_source_line	Transform jupyter magic commands to valid python code.
create_intermediate_py_file	Parse a notebook at `notebook_path` and saves a parsed version.
create_temp_path	Create the path for a parsed jupyter notebook.
get_notebook_code_cells	Parse a notebook and returns a Tuple.
get_rel_paths	Transform *file_paths* in a list of paths relative to *base_path*.
ignore_cell	Return True if the cell isn't a code cell or is empty.
is_parent_dir	Check if a given dir *parent_dir* is parent directory of *path*.
map_intermediate_to_input	Map intermediate file lines to notebook cell and line.
read_notebook_to_cells	Parse the notebook at `notebook_path` as Json and returns a list of notebook cells.

convert_source_line

convert_source_line(*source_line: str*) → str

Transform jupyter magic commands to valid python code.

This utilizes `nbconvert.filters.ipython2python`.

Parameters
source_line (*str*) – Single line of source code.

Returns
Valid python code, as string, even if it was a jupyter magic line.

Return type
str

create_intermediate_py_file

create_intermediate_py_file(*notebook_path: str*, *intermediate_dir_base_path: str*) → tuple[str, InputLineMapping]

Parse a notebook at `notebook_path` and saves a parsed version.

The corresponding position is relative to `intermediate_dir_base_path`.

Parameters

- **notebook_path** (*str*) – Path to a notebook.
- **intermediate_dir_base_path** (*str*) – Path pointing to the position the parsed notebook will be saved to.

Returns
(`intermediate_file_path`, `input_line_mapping`) Where `intermediate_file_path` is the path the parsed notebook was written to. If there was an error parsing the file the `intermediate_file_path` will be "". `input_line_mapping` is a dict which has the keys 'input_names' and 'code_lines'. `code_lines` is a List of the code cells `In[\d\*]` names and `code_lines` is the corresponding line in the parsed notebook.

Return type
tuple[str, InputLineMapping]

See also:

read_notebook_to_cells, *get_notebook_code_cells*, *create_temp_path*

Warns
InvalidNotebookWarning – If the notebook couldn't be parsed.

create_temp_path

create_temp_path(*notebook_path: str*, *temp_base_path: str*) → str

Create the path for a parsed jupyter notebook.

The path has the same relative position to `temp_base_path` as `notebook_path` has to `os.curdir`. If that would lead out of the `temp_base_path`, the path will point to a file at the root of `temp_base_path`, which has the same filename as the file at `notebook_path` has.

Parameters

- **notebook_path** (*str*) – Path to a notebook.
- **temp_base_path** (*str*) – Base path of a temporary folder, the new path should have the same relative position to as `notebook_path` has to `os.curdir`

Returns
Path to the temporary file which should be created.

Return type
str

get_notebook_code_cells

get_notebook_code_cells(*notebook_path: str*) → tuple[bool, list[NotebookCell]]

Parse a notebook and returns a Tuple.

The first entry being a bool which indicates if juypter magic was used and the second entry is a List of all code cells, as their dict representation.

Parameters
notebook_path (*str*) – Path to a notebook.

Returns
(`uses_get_ipython`, `notebook_cells`), where `uses_get_ipython` is a bool, which is `True` if any cell contained jupyter magic and `notebook_cells` is a List of all code cells dict representation.

Return type
tuple[bool, list[NotebookCell]]

See also:

read_notebook_to_cells

Warns
InvalidNotebookWarning – If the notebook couldn't be parsed.

get_rel_paths

get_rel_paths(*file_paths: list[str]*, *base_path: str*) → list[str]

Transform *file_paths* in a list of paths relative to *base_path*.

Parameters

- **file_paths** (*list[str]*) – List of file paths.
- **base_path** (*str*) – Path *file_paths* should be relative to.

Returns
List of *file_paths* relative to *base_path*

Return type
list[str]

ignore_cell

ignore_cell(*notebook_cell: Dict[str, Any]*) → bool

Return True if the cell isn't a code cell or is empty.

Parameters
notebook_cell (*NotebookCell*) – Dict representation of a notebook cell as parsed from JSON.

Returns
Whether cell should be ignored or not.

Return type
bool

is_parent_dir

is_parent_dir(*parent_dir: str*, *path: str*) → bool

Check if a given dir *parent_dir* is parent directory of *path*.

Parameters

- **parent_dir** (*str*) – Path to the directory, which should be checked if it is a parent directory of *path*.
- **path** (*str*) – Path to a file or directory, which should be checked if it is inside of *parent_dir*..

Returns
Weather or not 'path' is inside of 'parent_dir'.

Return type
bool

map_intermediate_to_input

map_intermediate_to_input(*input_line_mapping: InputLineMapping*, *line_number: int*) → tuple[*CellId*, int]

Map intermediate file lines to notebook cell and line.

Maps the line at *line_number* to the corresponding code cell (*input_cell_name*) and line number in the code cell (*input_cell_line_number*)

Parameters

- **input_line_mapping** (*InputLineMapping*) – Dict containing lists of input cell names and their line in the intermediate file.
- **line_number** (*int*) – Line in the intermediate py file.

Returns
Input cell ID and corresponding line in that cell (input_id, input_cell_line_number)

Return type
tuple[*CellId*, int]

See also:

create_intermediate_py_file

read_notebook_to_cells

read_notebook_to_cells(*notebook_path: str*) → list[NotebookCell]

Parse the notebook at `notebook_path` as Json and returns a list of notebook cells.

Parameters
notebook_path (*str*) – Path to a notebook.

Returns
List of notebook cells if the notebook was parsed successfully or an empty list if the *.ipynb file couldn't be parsed.

Return type
list[NotebookCell]

Warns
InvalidNotebookWarning – If the notebook couldn't be parsed.

Classes

Summary

NotebookParser	Main parsing class for notebooks.

NotebookParser

class NotebookParser(*original_notebook_paths: list[str] | None = None*)

Main parsing class for notebooks.

`NotebookParser` utilizes that instance and class attributes are separated and class attributes allow sharing of information across instances. This is used to realize the mapping of checked parsed notebooks back to their original files.

Initialize NotebookParser.

Initializing an instance of the class will save `original_notebook_paths`, which is a List of paths to notebooks, to the class attributes, which can be accessed by all instances or all modules that know about the class. If `original_notebook_paths` isn't provided, the class attributes will stay as it was.

Parameters

original_notebook_paths (*List[str], optional*) – List of paths to notebooks, by default None

Attributes Summary

`input_line_mappings`	List of input_line_mapping
`intermediate_py_file_paths`	List of paths to the parsed Notebooks
`original_notebook_paths`	List of paths to the original Notebooks
`temp_path`	Path of the temp folder the parsed notebooks were saved in

Methods Summary

clean_up	Delete the created temporary directory if it exists and resets all class attributes.
create_intermediate_py_file_paths	Create intermediate files needed for analysis.
get_mappings	Return the mapping information needed to generate error messages.

clean_up

static NotebookParser.**clean_up**() → None

Delete the created temporary directory if it exists and resets all class attributes.

create_intermediate_py_file_paths

NotebookParser.**create_intermediate_py_file_paths**() → None

Create intermediate files needed for analysis.

Parses all notebooks provided by `self.original_notebook_paths` and saves them to a temporary directory, if `original_notebook_paths`, was provided at initialization.

get_mappings

static NotebookParser.**get_mappings**() → Iterator[tuple[str, str, InputLineMapping]]

Return the mapping information needed to generate error messages.

The message corresponds to the original notebook and not the actually checked parsed one.

Returns

(`original_notebook_paths`, `intermediate_py_file_paths`, `input_line_mappings`) `original_notebook_paths` is the relative path of the tested notebook. `intermediate_py_file_paths` is the absolute path of the checked notebook. And `input_line_mapping` is a dict of information about which input in the original notebook, is in what line in the corresponding pared notebook.

Return type

Iterator[tuple[str, str, InputLineMapping]]

See also:

`input_line_mapping`, *create_intermediate_py_file*

Methods Documentation

static clean_up() → None

Delete the created temporary directory if it exists and resets all class attributes.

create_intermediate_py_file_paths() → None

Create intermediate files needed for analysis.

Parses all notebooks provided by `self.original_notebook_paths` and saves them to a temporary directory, if `original_notebook_paths`, was provided at initialization.

static get_mappings() → Iterator[tuple[str, str, InputLineMapping]]

Return the mapping information needed to generate error messages.

The message corresponds to the original notebook and not the actually checked parsed one.

Returns

(`original_notebook_paths`, `intermediate_py_file_paths`, `input_line_mappings`) `original_notebook_paths` is the relative path of the tested notebook. `intermediate_py_file_paths` is the absolute path of the checked notebook. And `input_line_mapping` is a dict of information about which input in the original notebook, is in what line in the corresponding pared notebook.

Return type

Iterator[tuple[str, str, InputLineMapping]]

See also:

`input_line_mapping`, *create_intermediate_py_file*

Exceptions

Exception Summary

InvalidNotebookWarning	Warning that is given when a jupyter notebook can't be parsed as JSON.

InvalidNotebookWarning

exception InvalidNotebookWarning(*notebook_path: str*)

Warning that is given when a jupyter notebook can't be parsed as JSON.

Initialize InvalidNotebookWarning.

Parameters
notebook_path (*str*) – Path to a notebook

Classes

Summary

CellId	Container to hold information to identify a cell.

CellId

class CellId(*input_nr: str*, *code_cell_nr: int*, *total_cell_nr: int*)

Container to hold information to identify a cell.

The information are: * `input_nr`

Execution count, " " for not executed cells

- **code_cell_nr**
  Count of the code cell starting at 1, ignoring raw and markdown cells
- **total_cell_nr**
  Total count of the cell starting at 1, considering raw and markdown cells.

Create new instance of CellId(input_nr, code_cell_nr, total_cell_nr)

Attributes Summary

code_cell_nr	Alias for field number 1
input_nr	Alias for field number 0
total_cell_nr	Alias for field number 2

Methods Summary

count	Return number of occurrences of value.
index	Return first index of value.

count

CellId.**count**(*value*, /)

Return number of occurrences of value.

index

CellId.**index**(*value*, *start=0*, *stop=9223372036854775807*, /)

Return first index of value.

Raises ValueError if the value is not present.

Methods Documentation

count(*value*, /)

Return number of occurrences of value.

index(*value*, *start=0*, *stop=9223372036854775807*, /)

Return first index of value.

Raises ValueError if the value is not present.

5.1.3 Functions

Summary

save_cast_int	Cast version string to tuple, in a save manner.

save_cast_int

save_cast_int(*int_str: str*) → int

Cast version string to tuple, in a save manner.

This is needed so the version number of prereleases (i.e. 3.8.0rc1) don't not throw exceptions.

Parameters
int_str (*str*) – String which should represent a number.

Returns
Int representation of int_str

Return type
int

CHAPTER

SIX

CONTRIBUTING

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

6.1 Types of Contributions

6.1.1 Report Bugs

Report bugs at https://github.com/s-weigand/flake8-nb/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

6.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" and "help wanted" is open to whoever wants to implement it.

6.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with "enhancement" and "help wanted" is open to whoever wants to implement it.

6.1.4 Write Documentation

flake8-nb could always use more documentation, whether as part of the official flake8-nb docs, in docstrings, or even on the web in blog posts, articles, and such.

6.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/s-weigand/flake8-nb/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

6.2 Get Started!

Ready to contribute? Here's how to set up `flake8_nb` for local development.

1. Fork the `flake8_nb` repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/flake8_nb.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv flake8_nb
$ cd flake8_nb/
$ pip install -r requirements_dev.txt
$ pip install -e .
```

4. Install the `pre-commit` hooks, for quality assurance:

```
$ pre-commit install
```

5. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

6. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ tox
```

   To get flake8 and tox, just pip install them into your virtualenv.

7. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

8. Submit a pull request through the GitHub website.

6.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 3.7, 3.8, 3.9 and 3.10. Check https://github.com/s-weigand/flake8-nb/actions and make sure that the tests pass for all supported Python versions.

6.4 Tips

To run a subset of tests:

```
$ pytest tests.test_flake8_nb
```

6.5 Deploying

A reminder for the maintainers on how to deploy. Make sure all your changes are committed (including an entry in HISTORY.rst). Then run:

```
$ bump2version patch # possible: major / minor / patch
$ git push
$ git push --tags
```

Travis will then deploy to PyPI if tests pass.

CHAPTER

SEVEN

CREDITS

Many thanks go to the creators and contributors of flake8. Which inspired me to write this hack on top of it and supplies nearly all of the functionality.

This packages skeleton was created with Cookiecutter and the audreyr/cookiecutter-pypackage project template.

The idea to use cell tags was inspired by the use of cell tags in nbval.

CHAPTER

EIGHT

CHANGELOG

8.1 0.5.3 (2023-03-28)

- Official python 3.11 support #291
- Fixed bug with pre-commit-ci due to read-only filesystem by @lkeegan in #290

8.2 0.5.2 (2022-08-17)

- Fix config file discovery with flake8>=5.0.0 #255

8.3 0.5.1 (2022-08-16)

- Fix config discovery with flake8>=5.0.0 #251

8.4 0.5.0 (2022-08-15)

- Drop support for `flake8<3.8.0` #240
- Set max supported version of `flake8` to be `<5.0.5` #240
- Enable calling `flake8_nb` as python module #240

8.5 0.4.0 (2022-02-21)

- Drop official python 3.6 support

8.6 0.3.1 (2021-10-19)

- Set max supported version of `flake8` to be `<4.0.2`
- Added official Python 3.10 support and tests

8.7 0.3.0 (2020-05-16)

- Set max supported version of `flake8` to be `<3.9.3`
- Report formatting is configurable via `--notebook-cell-format` option with formatting options `nb_path`, `exec_count`, `code_cell_count` and `total_cell_count`.

8.8 0.2.7 (2020-04-16)

- Set max supported version of `flake8` to be `<3.9.2`

8.9 0.2.6 (2020-03-21)

- Set max supported version of `flake8` to be `<3.9.1`

8.10 0.2.5 (2020-10-06)

- Added official Python 3.9 support and tests

8.11 0.2.4 (2020-10-04)

- Set max supported version of `flake8` to be `<3.8.5`

8.12 0.2.3 (2020-10-02)

- Fixed pre-commit hook file association so it support python and juypter notebooks

8.13 0.2.1 (2020-08-11)

- Forced uft8 encoding when reading notebooks, this prevents errors on windows when console codepage is assumed

8.14 0.2.0 (2020-07-18)

- Added pre-commit hook (#47)

8.15 0.1.8 (2020-06-09)

- Set max supported version of `flake8` to be `<=3.8.3`

8.16 0.1.7 (2020-05-25)

- Set max supported version of `flake8` to be `<=3.8.2`

8.17 0.1.6 (2020-05-20)

- Set max supported version of `flake8` to be `<=3.8.1`
- Fixed bug with `--exclude` option

8.18 0.1.4 (2020-01-01)

- Set max supported version of `flake8` to be `<3.8.0`, to prevent breaking due to changes of `flake8`'s inner workings.

8.19 0.1.3 (2019-11-13)

- Added official Python 3.8 support and tests

8.20 0.1.2 (2019-10-29)

- Fixed compatibility with `flake8==3.7.9`

8.21 0.1.1 (2019-10-24)

- Added console-script 'flake8-nb' as an alias for 'flake8_nb'

8.22 0.1.0 (2019-10-22)

- First release on PyPI.

CHAPTER

NINE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

f

INDEX

M

N

P

R

S

U

W