
nitpick Documentation

Release 0.35.0

W. Augusto Andreoli

Jan 25, 2024

CONTENTS:

1	Quickstart	3
1.1	Install	3
1.2	Run	3
1.3	Run as a pre-commit hook	4
1.4	Run as a MegaLinter plugin	5
1.5	Modify files directly	5
2	Styles	7
2.1	The style file	7
2.2	Configure your own style	8
2.3	Default search order for a style	8
2.4	Style file syntax	8
2.5	Special configurations	9
2.6	Breaking changes	10
3	Configuration	11
3.1	Remote style	11
3.2	Style inside Python package	12
3.3	Cache	13
3.4	Local style	14
3.5	Multiple styles	14
3.6	Override a remote style	14
4	Library (Presets)	15
4.1	any	15
4.2	javascript	15
4.3	kotlin	15
4.4	markdown	16
4.5	presets	16
4.6	proto	16
4.7	python	16
4.8	shell	17
4.9	toml	17
5	The [nitpick] table	19
5.1	Minimum version	19
5.2	[nitpick.files]	19
5.3	[nitpick.styles]	20
6	Command-line interface	21
6.1	Suggest styles based on project files	21

6.2	Main options	22
6.3	fix: Modify files directly	22
6.4	check: Don't modify, just print the differences	23
6.5	ls: List configures files	23
6.6	init: Initialise a configuration file	23
7	Flake8 plugin	25
7.1	Pre-commit hook and flake8	25
7.2	Root dir of the project	26
7.3	Main Python file	27
8	Plugins	29
8.1	INI files	29
8.2	JSON files	29
8.3	Text files	29
8.4	TOML files	30
8.5	YAML files	30
9	Troubleshooting	31
9.1	Crash on multi-threading	31
9.2	ModuleNotFoundError: No module named 'nitpick.plugins.XXX'	31
9.3	Executable .tox/lint/bin/pylint not found	32
9.4	Missing rev key when using the default pre-commit styles	32
10	Contributing	33
10.1	Bug reports or feature requests	33
10.2	Documentation improvements	33
10.3	Development	33
11	Authors	35
12	nitpick	37
12.1	nitpick package	37
13	Indices and tables	129
14	To Do List	131
	Python Module Index	133
	Index	135

Command-line tool and [flake8](#) plugin to enforce the same settings across multiple language-independent projects.

Useful if you maintain multiple projects and are tired of copying/pasting the same INI/TOML/YAML/JSON keys and values over and over, in all of them.

The CLI now has a `nitpick fix` command that modifies configuration files directly (pretty much like [black](#) and [isort](#) do with Python files). See [Command-line interface](#) for more info.

Many more features are planned for the future, check [the roadmap](#).

Note: This project is still a work in progress, so the API is not fully defined:

- [The style file](#) syntax might have changes before the 1.0 stable release;
 - The numbers in the NIP* error codes might change; don't fully rely on them;
 - See also [Breaking changes](#).
-

QUICKSTART

1.1 Install

Install in an isolated environment with `pipx`:

```
# Latest PyPI release
pipx install nitpick

# Development branch from GitHub
pipx install git+https://github.com/andreoliwa/nitpick
```

On macOS/Linux, install the latest release with `Homebrew`:

```
brew install andreoliwa/formulae/nitpick

# Development branch from GitHub
brew install andreoliwa/formulae/nitpick --HEAD
```

On Arch Linux, install with `yay`:

```
yay -Syu nitpick
```

Add to your project with `Poetry`:

```
poetry add --dev nitpick
```

Or install it with `pip`:

```
pip install -U nitpick
```

1.2 Run

`Nitpick` will fail if no style is explicitly configured. Run this command to download and use the opinionated `default style file`:

```
nitpick init
```

You can use it as a template to *Configure your own style*.

To fix and modify your files directly:

```
nitpick fix
```

To check for errors only:

```
nitpick check
```

Nitpick is also a [flake8](#) plugin, so you can run this on a project with at least one Python (.py) file:

```
flake8 .
```

1.3 Run as a pre-commit hook

If you use [pre-commit](#) on your project, add this to the `.pre-commit-config.yaml` in your repository:

```
repos:
- repo: https://github.com/andreoliwa/nitpick
  rev: v0.35.0
  hooks:
  - id: nitpick-suggest
  - id: nitpick
```

There are 4 available hook IDs:

- `nitpick` and `nitpick-fix` both run the `nitpick fix` command;
- `nitpick-check` runs `nitpick check`;
- `nitpick-suggest` runs `nitpick init --fix --suggest`;

If you want to run [Nitpick](#) as a [flake8](#) plugin instead:

```
repos:
- repo: https://github.com/PyCQA/flake8
  rev: 4.0.1
  hooks:
  - id: flake8
    additional_dependencies: [nitpick]
```

To install the `pre-commit` and `commit-msg` Git hooks:

```
pre-commit install --install-hooks
pre-commit install -t commit-msg
```

To start checking all your code against the default rules:

```
pre-commit run --all-files
```


1.4 Run as a MegaLinter plugin

If you use [MegaLinter](#) you can run Nitpick as a plugin. Add the following two entries to your `.mega-linter.yml` configuration file:

PLUGINS:

- <https://raw.githubusercontent.com/andreoliwa/nitpick/v0.35.0/mega-linter-plugin-nitpick/nitpick.megalinter-descriptor.yml>

ENABLE_LINTERS:

- NITPICK

1.5 Modify files directly

Nitpick includes a CLI to apply your style and modify the configuration files directly:

```
nitpick fix
```

Read more details here: [Command-line interface](#).

2.1 The style file

A “Nitpick code style” is a TOML file with the settings that should be present in config files from other tools.

Example of a style:

```
["pyproject.toml".tool.black]
line-length = 120

["pyproject.toml".tool.poetry.dev-dependencies]
pylint = "*"

["setup.cfg".flake8]
ignore = "D107,D202,D203,D401"
max-line-length = 120
inline-quotes = "double"

["setup.cfg".isort]
line_length = 120
multi_line_output = 3
include_trailing_comma = true
force_grid_wrap = 0
combine_as_imports = true
```

This example style will assert that:

- ... `black`, `isort` and `flake8` have a line length of 120;
- ... `flake8` and `isort` are configured with the above options in `setup.cfg`;
- ... `Pylint` is present as a `Poetry` dev dependency in `pyproject.toml`.

2.2 Configure your own style

After creating your own TOML file with your style, add it to your `pyproject.toml` file. See *Configuration* for details.

You can also check *the style library*, and use the styles to build your own.

2.3 Default search order for a style

1. A file or URL configured in the `pyproject.toml` file, `[tool.nitpick]` section, `style` key, as described in *Configuration*.
2. Any `nitpick-style.toml` file found in the current directory (the one in which `flake8` runs from) or above.
3. If no style is found, then the [default style file](#) from GitHub is used.

2.4 Style file syntax

A style file contains basically the configuration options you want to enforce in all your projects.

They are just the config to the tool, prefixed with the name of the config file.

E.g.: To configure the `black` formatter with a line length of 120, you use this in your `pyproject.toml`:

```
[tool.black]
line-length = 120
```

To enforce that all your projects use this same line length, add this to your `nitpick-style.toml` file:

```
["pyproject.toml"].tool.black
line-length = 120
```

It's the same exact section/key, just prefixed with the config file name ("`pyproject.toml`".)

The same works for `setup.cfg`.

To [configure mypy](#) to ignore missing imports in your project, this is needed on `setup.cfg`:

```
[mypy]
ignore_missing_imports = true
```

To enforce all your projects to ignore missing imports, add this to your `nitpick-style.toml` file:

```
["setup.cfg"].mypy
ignore_missing_imports = true
```

2.5 Special configurations

2.5.1 Comparing elements on lists

Note: At the moment, this feature only works on [:ref: the YAML plugin <yamlplugin>`_](#).

On YAML files, a list (called a “sequence” in the YAML spec) can contain different elements:

- Scalar values like int, float and string;
- Dictionaries or objects with key/value pairs (called “mappings” in the YAML spec)
- Other lists (sequences).

The default behaviour for all lists: when applying a style, Nitpick searches if the element already exists in the list; if not found, the element is appended to the end of list.

1. With scalar values, Nitpick compares the elements by their exact value. Strings are compared in a case-sensitive manner, and spaces are considered.
2. Dicts are compared by a “hash” of their key/value pairs.

On lists containing dicts, it is possible to define a custom “list key” used to search for an element, overriding the default compare mechanism above.

If an element is found by its key, the other key/values are compared and Nitpick applies the difference. If the key doesn’t exist, the new dict is appended to the end of the list.

Some files have a predefined configuration:

1. On `.pre-commit-config.yaml`, repos are searched by their hook IDs.
2. On GitHub Workflow YAML files, steps are searched by their names.

You can define your own list keys for your YAML files (or override the predefined configs above) by using `__list_keys` on your style:

```
["path/from/root/to/your/config.yaml".__list_keys]
"<list expression>" = "<search key expression>"
```

`<list expression>`:

- a dotted path to a list, e.g. `repos`, `path.to.some.key`; or
- a pattern containing wildcards (`?` and `*`). For example, `jobs.*.steps`: all jobs containing steps will use the same `<search key expression>`.

`<search key expression>`:

- a key from the dict inside the list, e.g. `name`; or
- a parent key and its child key separated by a dot, e.g. `hooks.id`.

Warning: For now, only two-level nesting is possible: parent and child keys.

If you have suggestions for predefined list keys for other popular YAML files not covered by Nitpick (e.g.: GitLab CI config), feel free to [create an issue](#) or submit a pull request.

2.6 Breaking changes

Warning: Below are the breaking changes in the style before the API is stable. If your style was working in a previous version and now it's not, check below.

2.6.1 `missing_message` key was removed

`missing_message` was removed. Use `[nitpick.files.present]` now.

Before:

```
[nitpick.files."pyproject.toml"]  
missing_message = "Install poetry and run 'poetry init' to create it"
```

Now:

```
[nitpick.files.present]  
"pyproject.toml" = "Install poetry and run 'poetry init' to create it"
```

CONFIGURATION

The style file for your project should be configured in the `[tool.nitpick]` table of the configuration file.

Possible configuration files (in order of precedence):

1. `.nitpick.toml`
2. `pyproject.toml`

The first file found will be used; the other files will be ignored.

If no style is configured, Nitpick will fail with an error message.

Run `nitpick init` to create a config file (*init: Initialise a configuration file*).

To configure your own style, you can either use `nitpick init`:

```
$ nitpick init /path/to/your-style-file.toml
```

or edit your configuration file and set the `style` option:

```
[tool.nitpick]
style = "/path/to/your-style-file.toml"
```

You can set `style` with any local file or URL.

3.1 Remote style

Use the URL of the remote file.

If it's hosted on GitHub, use any of the following formats:

GitHub URL scheme (`github://` or `gh://`) pinned to a specific version:

```
[tool.nitpick]
style = "github://andreoliwa/nitpick@v0.35.0/nitpick-style.toml"
# or
style = "gh://andreoliwa/nitpick@v0.35.0/nitpick-style.toml"
```

The `@` syntax is used to get a Git reference (commit, tag, branch). It is similar to the syntax used by `pip` and `pipx`:

- `pip install` - VCS Support - Git;
- `pypa/pipx`: Installing from Source Control.

If no Git reference is provided, the default GitHub branch will be used (for Nitpick, it's `develop`):

```
[tool.nitpick]
style = "github://andreoliwa/nitpick/nitpick-style.toml"
# or
style = "gh://andreoliwa/nitpick/nitpick-style.toml"

# It has the same effect as providing the default branch explicitly:
style = "github://andreoliwa/nitpick@develop/nitpick-style.toml"
# or
style = "gh://andreoliwa/nitpick@develop/nitpick-style.toml"
```

A regular GitHub URL also works. The corresponding raw URL will be used.

```
[tool.nitpick]
style = "https://github.com/andreoliwa/nitpick/blob/v0.35.0/nitpick-style.toml"
```

Or use the raw GitHub URL directly:

```
[tool.nitpick]
style = "https://raw.githubusercontent.com/andreoliwa/nitpick/v0.35.0/nitpick-style.toml"
```

You can also use the raw URL of a GitHub Gist:

```
[tool.nitpick]
style = "https://gist.githubusercontent.com/andreoliwa/f4fccf4e3e83a3228e8422c01a48be61/
↪raw/ff3447bddfc5a8665538ddf9c250734e7a38eabb/remote-style.toml"
```

If your style is on a private GitHub repo, you can provide the token directly on the URL. Or you can use an environment variable to avoid keeping secrets in plain text.

```
[tool.nitpick]
# A literal token
style = "github://p5iCG5AJuDgY@some-user/a-private-repo@some-branch/nitpick-style.toml"

# Or reading the secret value from the MY_AUTH_KEY env var
style = "github://$MY_AUTH_KEY@some-user/a-private-repo@some-branch/nitpick-style.toml"
```

Note: A literal token cannot start with a \$. All tokens must not contain any @ or : characters.

3.2 Style inside Python package

The style file can be fetched from an installed Python package.

Example of a use case: you create a custom flake8 extension and you also want to distribute a (versioned) Nitpick style bundled as a resource inside the Python package (check out this issue: [Get style file from python package · Issue #202](#)).

Python package URL scheme is `pypackage://` or `py://`:

```
[tool.nitpick]
style = "pypackage://some_python_package/styles/nitpick-style.toml"
# or
style = "py://some_python_package/styles/nitpick-style.toml"
```


Thanks to @isac322 for this feature.

3.3 Cache

Remote styles can be cached to avoid unnecessary HTTP requests. The cache can be configured with the cache key; see the examples below.

By default, remote styles will be cached for **one hour**. This default will also be used if the cache key has an invalid value.

3.3.1 Expiring after a predefined time

The cache can be set to expire after a defined time unit. Use the format `cache = "<integer> <time unit>"`. *Time unit* can be one of these (plural or singular, it doesn't matter):

- `minutes / minute`
- `hours / hour`
- `days / day`
- `weeks / week`

To cache for 15 minutes:

```
[tool.nitpick]
style = "https://example.com/remote-style.toml"
cache = "15 minutes"
```

To cache for 1 day:

```
[tool.nitpick]
style = "https://example.com/remote-style.toml"
cache = "1 day"
```

3.3.2 Forever

With this option, once the style(s) are cached, they never expire.

```
[tool.nitpick]
style = "https://example.com/remote-style.toml"
cache = "forever"
```

3.3.3 Never

With this option, the cache is never used. The remote style file(s) are always looked-up and a HTTP request is always executed.

```
[tool.nitpick]
style = "https://example.com/remote-style.toml"
cache = "never"
```

3.3.4 Clearing

The cache files live in a subdirectory of your project: `/path/to/your/project/.cache/nitpick/`. To clear the cache, simply remove this directory.

3.4 Local style

Using a file in your home directory:

```
[tool.nitpick]
style = "~/some/path/to/another-style.toml"
```

Using a relative path from another project in your hard drive:

```
[tool.nitpick]
style = "../another-project/another-style.toml"
```

3.5 Multiple styles

You can also use multiple styles and mix local files and URLs.

Example of usage: the `[tool.nitpick]` table on Nitpick's own `pyproject.toml`.

```
[tool.nitpick]
style = [
    "/path/to/first.toml",
    "/another/path/to/second.toml",
    "https://example.com/on/the/web/third.toml"
]
```

Note: The order is important: each style will override any keys that might be set by the previous `.toml` file.

If a key is defined in more than one file, the value from the last file will prevail.

3.6 Override a remote style

You can use a remote style as a starting point, and override settings on your local style file.

Use `./` to indicate the local style:

```
[tool.nitpick]
style = [
    "https://example.com/on/the/web/remote-style.toml",
    "./my-local-style.toml",
]
```

For Windows users: even though the path separator is a backslash, use the example above as-is. The “dot-slash” is a convention for Nitpick to know this is a local style file.

LIBRARY (PRESETS)

If you want to *configure your own style*, those are the some styles you can reuse.

Many TOML configs below are used in the [default style file](#).

You can use these examples directly with their `py://` URL (see [Multiple styles](#)), or copy/paste the TOML into your own style file.

4.1 any

Style URL	Description
<code>py://nitpick/resources/any/codeclimate</code>	CodeClimate
<code>py://nitpick/resources/any/commitizen</code>	Commitizen (Python)
<code>py://nitpick/resources/any/commitlint</code>	commitlint
<code>py://nitpick/resources/any/editorconfig</code>	EditorConfig
<code>py://nitpick/resources/any/git-legal</code>	Git.legal - CodeClimate Community Edition
<code>py://nitpick/resources/any/pre-commit-hooks</code>	pre-commit hooks for any project
<code>py://nitpick/resources/any/prettier</code>	Prettier

4.2 javascript

Style URL	Description
<code>py://nitpick/resources/javascript/package-json</code>	package.json

4.3 kotlin

Style URL	Description
<code>py://nitpick/resources/kotlin/ktlint</code>	ktlint

4.4 markdown

Style URL	Description
py://nitpick/resources/markdown/markdownlint	Markdown lint

4.5 presets

Style URL	Description
py://nitpick/resources/presets/nitpick	Default style file for Nitpick

4.6 proto

Style URL	Description
py://nitpick/resources/proto/protolint	protolint (Protobuf linter)

4.7 python

Style URL	Description
py://nitpick/resources/python/310	Python 3.10
py://nitpick/resources/python/311	Python 3.11
py://nitpick/resources/python/312	Python 3.12
py://nitpick/resources/python/38	Python 3.8
py://nitpick/resources/python/39	Python 3.9
py://nitpick/resources/python/absent	Files that should not exist
py://nitpick/resources/python/autoflake	autoflake
py://nitpick/resources/python/bandit	Bandit
py://nitpick/resources/python/black	Black
py://nitpick/resources/python/flake8	Flake8
py://nitpick/resources/python/github-workflow	GitHub Workflow for Python
py://nitpick/resources/python/ipython	IPython
py://nitpick/resources/python/isort	isort
py://nitpick/resources/python/mypy	Mypy
py://nitpick/resources/python/poetry-editable	Poetry (editable projects; PEP 600 support)
py://nitpick/resources/python/poetry-venv	Poetry (virtualenv in project)
py://nitpick/resources/python/poetry	Poetry
py://nitpick/resources/python/pre-commit-hooks	pre-commit hooks for Python projects
py://nitpick/resources/python/pylint	Pylint
py://nitpick/resources/python/radon	Radon
py://nitpick/resources/python/readthedocs	Read the Docs
py://nitpick/resources/python/sonar-python	SonarQube Python plugin
py://nitpick/resources/python/tox	tox

4.8 shell

Style URL	Description
py://nitpick/resources/shell/bashate	bashate (code style for Bash)
py://nitpick/resources/shell/shellcheck	ShellCheck (static analysis for shell scripts)
py://nitpick/resources/shell/shfmt	shfmt (shell script formatter)

4.9 toml

Style URL	Description
py://nitpick/resources/toml/toml-sort	TOML sort

THE [NITPICK] TABLE

The [nitpick] table in *the style file* contains global settings for the style.

Those are settings that either don't belong to any specific config file, or can be applied to all config files.

5.1 Minimum version

Show an upgrade message to the developer if Nitpick's version is below `minimum_version`:

```
[nitpick]
minimum_version = "0.10.0"
```

5.2 [nitpick.files]

5.2.1 Files that should exist

To enforce that certain files should exist in the project, you can add them to the style file as a dictionary of “file name” and “extra message”.

Use an empty string to not display any extra message.

```
[nitpick.files.present]
".editorconfig" = ""
"CHANGELOG.md" = "A project should have a changelog"
```

5.2.2 Files that should be deleted

To enforce that certain files should not exist in the project, you can add them to the style file.

```
[nitpick.files.absent]
"some_file.txt" = "This is an optional extra string to display after the warning"
"another_file.env" = ""
```

Multiple files can be configured as above. The message is optional.

5.2.3 Comma separated values

On `setup.cfg`, some keys are lists of multiple values separated by commas, like `flake8.ignore`.

On the style file, it's possible to indicate which key/value pairs should be treated as multiple values instead of an exact string. Multiple keys can be added.

```
[nitpick.files."setup.cfg"]
comma_separated_values = ["flake8.ignore", "isort.some_key", "another_section.another_key
↪"]
```

5.3 [nitpick.styles]

Styles can include other styles. Just provide a list of styles to include.

Example of usage: Nitpick's default style.

```
[nitpick.styles]
include = ["styles/python38", "styles/poetry"]
```

The styles will be merged following the sequence in the list. The `.toml` extension for each referenced file can be omitted.

Relative references are resolved relative to the URI of the style document they are included in according to the [normal rules of RFC 3986](#).

E.g. for a style file located at `gh://$GITHUB_TOKEN@foo_dev/bar_project@branchname/styles/foobar.toml` the following strings all reference the exact same canonical location to include:

```
[nitpick.styles]
include = [
  "foobar.toml",
  "../styles/foobar.toml",
  "/bar_project@branchname/styles/foobar.toml",
  "//$GITHUB_TOKEN@foo_dev/bar_project@branchname/styles/foobar.toml",
]
```

For style files on the local filesystem, the canonical path (after symbolic links have been resolved) of the style file is used as the base.

If a key/value pair appears in more than one sub-style, it will be overridden; the last declared key/pair will prevail.

COMMAND-LINE INTERFACE

Nitpick has a CLI command to fix files automatically.

1. It doesn't work for all the plugins yet. Currently, it works for:
 - *INI files* (like `setup.cfg`, `tox.ini`, `.editorconfig`, `.pylintrc`, and any other `.ini`)
 - *TOML files*
2. It tries to preserve the comments and the formatting of the original file.
3. Some changes still have to be done manually; Nitpick cannot guess how to make certain changes automatically.
4. Run `nitpick fix` to modify files directly, or `nitpick check` to only display the violations.
5. The `flake8` plugin only checks the files and doesn't make changes. This is the default for now; once the CLI becomes more stable, the "fix mode" will become the default.
6. The output format aims to follow `pycodestyle (pep8) default output format`.
7. You can set options through environment variables using the format `NITPICK_command_option`. E.e: `NITPICK_CHECK_VERBOSE=3 nitpick check`. For more details on how this works, see the [Click documentation about CLI options](#).

If you use Git, you can review the files before committing.

The available commands are described below.

6.1 Suggest styles based on project files

You can use the `nitpick init --suggest` command to suggest styles based on the files in your project.

Nitpick will scan the files starting from the project root, skipping the ones ignored by Git, and suggest the styles that match the file types.

By default, it will suggest the styles from the built-in library. You can use the `--library` option to scan a custom directory with styles.

The directory structure of your custom library should be the same as the built-in library: `<root_library_dir>/<file_type>/<your_style_name>.toml`.

- `file_type` can be any tag reported by the [pre-commit/identify library](#) (e.g. `python`, `yaml`, `markdown`, etc.);
- use the special type `any` for styles that apply to all file types (e.g. `trim trailing whitespace`).

The `--suggest` command will display what would be changed. To apply these changes and autofix the config, run with the --fix option.`

To ignore styles, add them to the `[tool.nitpick] ignore_styles` key on the config file.

Nitpick will consider the following Git ignore files:

- `.gitignore` on the project root;
- a global ignore file configured by `git config --get core.excludesFile`.

6.2 Main options

```
Usage: nitpick [OPTIONS] COMMAND [ARGS]...
```

Enforce the same settings across multiple language-independent projects.

Options:

```
-p, --project DIRECTORY Path to project root
--offline                Offline mode: no style will be downloaded (no HTTP
                        requests at all)
--version                Show the version and exit.
--help                  Show this message and exit.
```

Commands:

```
check Don't modify files, just print the differences.
fix    Fix files, modifying them directly.
init   Create or update the [tool.nitpick] table in the configuration...
ls     List of files configured in the Nitpick style.
```

6.3 fix: Modify files directly

At the end of execution, this command displays:

- the number of fixed violations;
- the number of violations that have to be changed manually.

```
Usage: nitpick fix [OPTIONS] [FILES]...
```

Fix files, modifying them directly.

You can use partial **and** multiple file names **in** the FILES argument.

Options:

```
-v, --verbose Increase logging verbosity (-v = INFO, -vv = DEBUG)
--help        Show this message and exit.
```

6.4 check: Don't modify, just print the differences

Usage: nitpick check [OPTIONS] [FILES]...

Don't modify files, just print the differences.

Return code 0 means nothing would change. Return code 1 means some files would be modified. You can use partial **and** multiple file names **in** the FILES argument.

Options:

-v, --verbose Increase logging verbosity (-v = INFO, -vv = DEBUG)
 --help Show this message **and** exit.

6.5 ls: List configures files

Usage: nitpick ls [OPTIONS] [FILES]...

List of files configured **in** the Nitpick style.

Display existing files **in** green **and** absent files **in** red. You can use partial **and** multiple file names **in** the FILES argument.

Options:

--help Show this message **and** exit.

6.6 init: Initialise a configuration file

Usage: nitpick init [OPTIONS] [STYLE_URLS]...

Create **or** update the [tool.nitpick] table **in** the configuration file.

Options:

-f, --fix Autofix the files changed by the command;
 otherwise, just **print** what would be done
 -s, --suggest Suggest styles based on the extension of project
 files (skipping Git ignored files)
 -l, --library DIRECTORY Library **dir** to scan **for** style files (implies
 --suggest); **if not** provided, uses the built-**in**
 style library
 --help Show this message **and** exit.

Note: Try running Nitpick with the new *Command-line interface* instead of a flake8 plugin.

In the future, there are plans to **make flake8 an optional dependency**.

FLAKE8 PLUGIN

Nitpick is not a proper `flake8` plugin; it piggybacks on `flake8`'s messaging system though.

Flake8 lints Python files; Nitpick “lints” configuration (text) files instead.

To act like a `flake8` plugin, Nitpick does the following:

1. Find *any Python file in your project*;
2. Use the first Python file found and ignore other Python files in the project.
3. Check the style file and compare with the configuration/text files.
4. Report violations on behalf of that Python file, and not on the configuration file that's actually wrong.

So, if you have a violation on `setup.cfg`, it will be reported like this:

```
./tasks.py:0:1: NIP323 File setup.cfg: [flake8]max-line-length is 80 but it should be 120
↳ like this:
[flake8]
max-line-length = 120
```

Notice the `tasks.py` at the beginning of the line, and not `setup.cfg`.

Note: To run Nitpick as a Flake8 plugin, the project must have *at least one* Python file*.

If your project is not a Python project, creating a `dummy.py` file on the root of the project is enough.

7.1 Pre-commit hook and flake8

Currently, the default pre-commit hook uses `flake8` in an unconventional and not recommended way.

It calls `flake8` directly:

```
flake8 --select=NIP
```

This current default pre-commit hook (called `nitpick`) is a placeholder for the future, when `flake8` will be only an optional dependency.

7.1.1 Why `always_run: true`?

This is intentional, because *Nitpick is not a conventional flake8 plugin*.

Since flake8 only lints Python files, the pre-commit hook will only run when a Python file is modified. It won't run when a config/text changes.

An example: suppose you're using a remote Nitpick style (like the style from WeMake).

At the moment, their style currently checks `setup.cfg` only.

Suppose they change or add an option on their `isort.toml` file.

If the nitpick pre-commit hook had `always_run: false` and `pass_filenames: true`, your local `setup.cfg` would only be verified:

1. If a Python file was changed.
2. If you ran `pre-commit run --all-files`.

So basically the pre-commit hook would be useless to guarantee that your config files would always match the remote style... which is precisely the purpose of Nitpick.

Note: To avoid this, use the [other pre-commit hooks](#), the ones that call the Nitpick CLI directly instead of running flake8.

7.2 Root dir of the project

You should run Nitpick in the *root dir* of your project.

A directory is considered a root dir if it contains one of the following files:

- `.pre-commit-config.yaml` (pre-commit)
- `pyproject.toml`
- `setup.py`
- `setup.cfg`
- `requirements*.txt`
- Pipfile (Pipenv)
- `tox.ini` (tox)
- `package.json` (JavaScript, NodeJS)
- `Cargo.*` (Rust)
- `go.mod`, `go.sum` (Golang)
- `app.py` and `wsgi.py` (Flask CLI)
- `autoapp.py` (Flask)
- `manage.py` (Django)

7.3 Main Python file

On the *root dir of the project* Nitpick searches for a main Python file. Every project must have at least one `*.py` file, otherwise `flake8` won't even work.

Those are the Python files that are considered:

- `setup.py`
- `app.py` and `wsgi.py` (Flask CLI)
- `autoapp.py`
- `manage.py`
- any `*.py` file

PLUGINS

Nitpick uses plugins to handle configuration files.

There are plans to add plugins that handle certain file types, specific files, and user plugins. Check [the roadmap](#).

Below are the currently included plugins.

8.1 INI files

Enforce configurations and autofix INI files.

Examples of `.ini` files handled by this plugin:

- `setup.cfg`
- `.editorconfig`
- `tox.ini`
- `.pylintrc`

Style examples enforcing values on INI files: [flake8 configuration](#).

8.2 JSON files

Enforce configurations and autofix JSON files.

Add the configurations for the file name you wish to check. Style example: [the default config for package.json](#).

8.3 Text files

Enforce configuration on text files.

To check if `some.txt` file contains the lines `abc` and `def` (in any order):

```
[[ "some.txt".contains ]]  
line = "abc"  
  
[[ "some.txt".contains ]]  
line = "def"
```

8.4 TOML files

Enforce configurations and autofix TOML files.

E.g.: `pyproject.toml` (PEP 518).

See also the `[tool.poetry]` section of the `pyproject.toml` file.

Style example: Python 3.8 version constraint. There are *many other examples here*.

8.5 YAML files

Enforce configurations and autofix YAML files.

- Example: `.pre-commit-config.yaml`.
- Style example: [the default pre-commit hooks](#).

Warning: The plugin tries to preserve comments in the YAML file by using the `ruamel.yaml` package. It works for most cases. If your comment was removed, place them in a different place of the file and try again. If it still doesn't work, please [report a bug](#).

Known issue: lists like `args` and `additional_dependencies` might be joined in a single line, and comments between items will be removed. Move your comments outside these lists, and they should be preserved.

Note: No validation of `.pre-commit-config.yaml` will be done anymore in this generic YAML plugin. Nitpick will not validate hooks and missing keys as it did before; it's not the purpose of this package.

TROUBLESHOOTING

9.1 Crash on multi-threading

On macOS, `flake8` might raise this error when calling `requests.get(url)`:

```
objc[93329]: +[__NSDate initialize] may have been in progress in another
↳ thread when fork() was called.
objc[93329]: +[__NSDate initialize] may have been in progress in another
↳ thread when fork() was called. We cannot safely call it or ignore it in the fork()
↳ child process. Crashing instead. Set a breakpoint on objc_initializeAfterForkError to
↳ debug.
```

To solve this issue, add this environment variable to `.bashrc` (or the initialization file for your favorite shell):

```
export OBJC_DISABLE_INITIALIZE_FORK_SAFETY=YES
```

Thanks to [this StackOverflow answer](#).

9.2 ModuleNotFoundError: No module named 'nitpick.plugins.XXX'

When upgrading to new versions, old plugins might be renamed in `setuptools` entry points.

But they might still be present in the `entry_points.txt` plugin metadata in your virtualenv.

```
$ rg nitpick.plugins.setup ~/Library/Caches/pypoetry/
/Users/john.doe/Library/Caches/pypoetry/virtualenvs/nitpick-UU_pZ5zs-py3.7/lib/python3.7/
↳ site-packages/nitpick-0.24.1.dist-info/entry_points.txt
11:setup_cfg=nitpick.plugins.setup_cfg
```

Remove and recreate the virtualenv; this should fix it.

During development, you can run `invoke clean --venv install --dry`. It will display the commands that would be executed; remove `--dry` to actually run them.

Read [this page](#) on how to install `Invoke`.

9.3 Executable `.tox/lint/bin/pylint` not found

You might get this error while running `make` locally.

1. Run `invoke lint` (or `tox -e lint` directly) to create this `tox` environment.
2. Run `make` again.

9.4 Missing `rev` key when using the default `pre-commit` styles

If you're using the default `pre-commit` styles, you might get this error:

```
An error has occurred: InvalidConfigError:
==> File .pre-commit-config.yaml
==> At Config()
==> At key: repos
==> At Repository(repo='https://github.com/PyCQA/bandit')
=====> Missing required key: rev
Check the log at /Users/your-name/.cache/pre-commit/pre-commit.log
```

This happens because the default styles don't have a `rev` key. Currently, this is not possible because the `pre-commit` plugin doesn't support it.

To solve this, you can run `pre-commit autoupdate` to update the styles to the latest version, as [recommended in the official docs](#).

For more details, [check out this comment on the GitHub issue](#).

CONTRIBUTING

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given. Check the [projects on GitHub](#), you might help coding a planned feature.

10.1 Bug reports or feature requests

- First, search the [GitHub issue tracker](#) to see if your bug/feature is already there.
- If nothing is found, just add a new issue and follow the instructions.

10.2 Documentation improvements

[Nitpick](#) could always use more documentation, whether as part of the official docs, in docstrings, or even on the web in blog posts, articles, and such.

10.3 Development

To set up [Nitpick](#) for local development:

1. Fork [Nitpick](#) (look for the “Fork” button).
2. Clone your fork locally:

```
cd ~/Code
git clone git@github.com:your_name_here/nitpick.git
cd nitpick
```

3. Install [Poetry](#) globally using the recommended way.
4. Install [Invoke](#). You can use [pipx](#) to install it globally: `pipx install invoke`.
5. Install dependencies and [pre-commit](#) hooks:

```
invoke install --hooks
```

6. Create a branch for local development:

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

Now you can make your changes locally.

7. When you're done making changes, run tests and checks locally with:

```
# Quick tests and checks
make
# Or use this to simulate a full CI build with tox
invoke ci-build
```

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

```
git add .

# For a feature:
git commit -m "feat: short description of your feature"
# For a bug fix:
git commit -m "fix: short description of what you fixed"

git push origin name-of-your-bugfix-or-feature
```

9. Submit a pull request through the GitHub website.

10.3.1 Commit convention

Nitpick follows [Conventional Commits](#)

No need to rebase the commits in your branch. If your pull request is accepted, all your commits will be squashed into a single one, and the commit message will be adjusted to follow the current standard.

10.3.2 Pull Request Guidelines

If you need some code review or feedback while you're developing the code, just make a draft pull request.

For merging, follow the checklist on the pull request template itself.

When running `invoke test`: if you don't have all the necessary Python versions available locally (needed by `tox`), you can rely on GitHub Workflows. [Tests will run](#) for each change you add in the pull request. It will be slower though...

CHAPTER
ELEVEN

AUTHORS

- W. Augusto Andreoli <andreoliwa@gmail.com>

12.1 nitpick package

Main module.

class nitpick.Nitpick

Bases: `object`

The Nitpick API.

configured_files(**partial_names*: *str*) → list[pathlib.Path]

List of files configured in the Nitpick style.

Filter only the selected partial names.

echo(*message*: *str*)

Echo a message on the terminal, with the relative path at the beginning.

enforce_present_absent(**partial_names*: *str*) → Iterator[Fuss]

Enforce files that should be present or absent.

Parameters

partial_names – Names of the files to enforce configs for.

Returns

Fuss generator.

enforce_style(**partial_names*: *str*, *autofix*=True) → Iterator[Fuss]

Read the merged style and enforce the rules in it.

1. Get all root keys from the merged style (every key is a filename, except “nitpick”).
2. For each file name, find the plugin(s) that can handle the file.

Parameters

- **partial_names** – Names of the files to enforce configs for.
- **autofix** – Flag to modify files, if the plugin supports it (default: True).

Returns

Fuss generator.

init(*project_root*: PathOrStr | None = None, *offline*: bool | None = None) → Nitpick

Initialize attributes of the singleton.

project: *Project*

run(**partial_names*: *str*, *autofix*=*False*) → *Iterator[Fuss]*

Run Nitpick.

Parameters

- **partial_names** – Names of the files to enforce configs for.
- **autofix** – Flag to modify files, if the plugin supports it (default: True).

Returns

Fuss generator.

classmethod singleton() → *Nitpick*

Return a single instance of the class.

12.1.1 Subpackages

nitpick.plugins package

Hook specifications used by Nitpick plugins.

Note: The hook specifications and the plugin classes are still experimental and considered as an internal API. They might change at any time; use at your own risk.

Submodules

nitpick.plugins.base module

Base class for file checkers.

class nitpick.plugins.base.**NitpickPlugin**(*info*: *FileInfo*, *expected_config*: *JsonDict*, *autofix*=*False*)

Bases: *object*

Base class for Nitpick plugins.

Parameters

- **data** – File information (project, path, tags).
- **expected_config** – Expected configuration for the file
- **autofix** – Flag to modify files, if the plugin supports it (default: True).

abstract enforce_rules() → *Iterator[Fuss]*

Enforce rules for this file. It must be overridden by inherited classes if needed.

entry_point() → *Iterator[Fuss]*

Entry point of the Nitpick plugin.

filename = ''

fixable: *bool* = **False**

Can this plugin modify its files directly? Are the files fixable?

identify_tags: `ClassVar = {}`

Which identify tags this `nitpick.plugins.base.NitpickPlugin` child recognises.

abstract property initial_contents: `str`

Suggested initial content when the file doesn't exist.

property nitpick_file_dict: `Dict[str, Any]`

Nitpick configuration for this file as a TOML dict, taken from the style file.

post_init()

Hook for plugin initialization after the instance was created.

The name mimics `__post_init__()` on dataclasses, without the magic double underscores: [Post-init processing](#)

predefined_special_config() \rightarrow `SpecialConfig`

Create a predefined special configuration for this plugin.

Each plugin can override this method.

skip_empty_suggestion = False

validation_schema: `Schema | None = None`

Nested validation field for this file, to be applied in runtime when the validation schema is rebuilt. Useful when you have a strict configuration for a file type (e.g. `nitpick.plugins.json.JsonPlugin`).

violation_base_code: `int = 0`

write_file(*file_exists: bool*) \rightarrow `Fuss | None`

Hook to write the new file when autofix mode is on.

Should be used by inherited classes.

write_initial_contents(*doc_class: type[nitpick.blender.BaseDoc]*, *expected_dict: dict | None = None*) \rightarrow `str`

Helper to write initial contents based on a format.

nitpick.plugins.info module

Info needed by the plugins.

class `nitpick.plugins.info.FileInfo`(*project: Project*, *path_from_root: str*, *tags: set[str] = <factory>*)

Bases: `object`

File information needed by the plugin.

classmethod `create`(*project: Project*, *path_from_root: str*) \rightarrow `FileInfo`

Clean the file name and get its tags.

path_from_root: `str`

project: `Project`

tags: `set[str]`

nitpick.plugins.ini module

INI files.

class nitpick.plugins.ini.**IniPlugin**(*info*: [FileInfo](#), *expected_config*: [JsonDict](#), *autofix*=False)

Bases: [NitpickPlugin](#)

Enforce configurations and autofix INI files.

Examples of `.ini` files handled by this plugin:

- `setup.cfg`
- `.editorconfig`
- `tox.ini`
- `.pylintrc`

Style examples enforcing values on INI files: [flake8 configuration](#).

add_options_before_space(*section*: *str*, *options*: *dict*) → None

Add new options before a blank line in the end of the section.

comma_separated_values: `set[str]`

compare_different_keys(*section*, *key*, *raw_actual*: *Any*, *raw_expected*: *Any*) → [Iterator\[Fuss\]](#)

Compare different keys, with special treatment when they are lists or numeric.

static contents_without_top_section(*multiline_text*: *str*) → *str*

Remove the temporary top section from multiline text, and keep the newline at the end of the file.

property current_sections: `set[str]`

Current sections of the `.ini` file, including updated sections.

dirty: `bool`

enforce_comma_separated_values(*section*, *key*, *raw_actual*: *Any*, *raw_expected*: *Any*) → [Iterator\[Fuss\]](#)

Enforce sections and keys with comma-separated values.

The values might contain spaces.

enforce_missing_sections() → [Iterator\[Fuss\]](#)

Enforce missing sections.

enforce_rules() → [Iterator\[Fuss\]](#)

Enforce rules on missing sections and missing key/value pairs in an INI file.

enforce_section(*section*: *str*) → [Iterator\[Fuss\]](#)

Enforce rules for a section.

entry_point() → [Iterator\[Fuss\]](#)

Entry point of the Nitpick plugin.

expected_config: [JsonDict](#)

property expected_sections: `set[str]`

Expected sections (from the style config).

file_path: [Path](#)

filename = ''

fixable: bool = True

Can this plugin modify its files directly? Are the files fixable?

static get_example_cfg(parser: ConfigParser) → str

Print an example of a config parser in a string instead of a file.

get_missing_output() → str

Get a missing output string example from the missing sections in an INI file.

identify_tags: ClassVar = {'editorconfig', 'ini'}

Which identify tags this `nitpick.plugins.base.NitpickPlugin` child recognises.

property initial_contents: str

Suggest the initial content for this missing file.

property missing_sections: set[str]

Missing sections.

property needs_top_section: bool

Return True if this .ini file needs a top section (e.g.: .editorconfig).

property nitpick_file_dict: Dict[str, Any]

Nitpick configuration for this file as a TOML dict, taken from the style file.

post_init()

Post initialization after the instance was created.

predefined_special_config() → SpecialConfig

Create a predefined special configuration for this plugin.

Each plugin can override this method.

show_missing_keys(section: str, values: list[tuple[str, Any]]) → Iterator[Fuss]

Show the keys that are not present in a section.

skip_empty_suggestion = False

updater: ConfigUpdater

validation_schema: Schema | None = None

Nested validation field for this file, to be applied in runtime when the validation schema is rebuilt. Useful when you have a strict configuration for a file type (e.g. `nitpick.plugins.json.JsonPlugin`).

violation_base_code: int = 320

write_file(file_exists: bool) → Fuss | None

Write the new file.

write_initial_contents(doc_class: type[nitpick.blender.BaseDoc], expected_dict: dict | None = None) → str

Helper to write initial contents based on a format.

class nitpick.plugins.ini.Violations(value, names=None, *, module=None, qualname=None, type=None, start=1, boundary=None)

Bases: `ViolationEnum`

Violations for this plugin.

```
INVALID_COMMA_SEPARATED_VALUES_SECTION = (325, ': invalid sections on  
comma_separated_values:')
```

```
MISSING_OPTION = (324, ': section [{section}] has some missing key/value pairs. Use  
this:')
```

```
MISSING_SECTIONS = (321, ' has some missing sections. Use this:')
```

```
MISSING_VALUES_IN_LIST = (322, ' has missing values in the {key!r} key. Include  
those values:')
```

```
OPTION_HAS_DIFFERENT_VALUE = (323, ': [{section}]{key} is {actual} but it should be  
like this:')
```

```
PARSING_ERROR = (326, ': parsing error ({cls}): {msg}')
```

```
TOP_SECTION_HAS_DIFFERENT_VALUE = (327, ': {key} is {actual} but it should be:')
```

```
TOP_SECTION_MISSING_OPTION = (328, ': top section has missing options. Use this:')
```

`nitpick.plugins.ini.can_handle(info: FileInfo) → type[NitpickPlugin] | None`

Handle INI files.

`nitpick.plugins.ini.plugin_class() → type[nitpick.plugins.base.NitpickPlugin]`

Handle INI files.

nitpick.plugins.json module

JSON files.

```
class nitpick.plugins.json.JsonFileSchema(*, only: Sequence[str] | AbstractSet[str] | None = None,  
exclude: Sequence[str] | AbstractSet[str] = (), many: bool =  
False, context: dict | None = None, load_only: Sequence[str] |  
AbstractSet[str] = (), dump_only: Sequence[str] |  
AbstractSet[str] = (), partial: bool | Sequence[str] |  
AbstractSet[str] | None = None, unknown: str | None = None)
```

Bases: [BaseNitpickSchema](#)

Validation schema for any JSON file added to the style.

class Meta

Bases: `object`

Options object for a Schema.

Example usage:

```
class Meta:  
    fields = ("id", "email", "date_created")  
    exclude = ("password", "secret_attribute")
```

Available options:

- `fields`: Tuple or list of fields to include in the serialized result.
- **additional**: Tuple or list of fields to include *in addition* to the explicitly declared fields. `additional` and `fields` are mutually-exclusive options.

- **include:** Dictionary of additional fields to include in the schema. It is usually better to define fields as class variables, but you may need to use this option, e.g., if your fields are Python keywords. May be an *OrderedDict*.
- **exclude:** Tuple or list of fields to exclude in the serialized result. Nested fields can be represented with dot delimiters.
- **dateformat:** Default format for *Date* <fields.Date> fields.
- **datetimeformat:** Default format for *DateTime* <fields.DateTime> fields.
- **timeformat:** Default format for *Time* <fields.Time> fields.
- **render_module:** Module to use for *loads* <Schema.loads> and *dumps* <Schema.dumps>. Defaults to *json* from the standard library.
- **ordered:** If *True*, output of *Schema.dump* will be a *collections.OrderedDict*.
- **index_errors:** If *True*, errors dictionaries will include the index of invalid items in a collection.
- **load_only:** Tuple or list of fields to exclude from serialized results.
- **dump_only:** Tuple or list of fields to exclude from deserialization
- **unknown:** Whether to exclude, include, or raise an error for unknown fields in the data. Use *EXCLUDE*, *INCLUDE* or *RAISE*.
- **register:** Whether to register the *Schema* with *marshmallow's* internal class registry. Must be *True* if you intend to refer to this *Schema* by class name in *Nested* fields. Only set this to *False* when memory usage is critical. Defaults to *True*.

OPTIONS_CLASS

alias of *SchemaOpts*

```
TYPE_MAPPING: Dict[type, Type[ma_fields.Field]] = {<class 'str'>: <class
'marshmallow.fields.String'>, <class 'bytes'>: <class 'marshmallow.fields.String'>,
<class 'datetime.datetime'>: <class 'marshmallow.fields.DateTime'>, <class
'float'>: <class 'marshmallow.fields.Float'>, <class 'bool'>: <class
'marshmallow.fields.Boolean'>, <class 'tuple'>: <class 'marshmallow.fields.Raw'>,
<class 'list'>: <class 'marshmallow.fields.Raw'>, <class 'set'>: <class
'marshmallow.fields.Raw'>, <class 'int'>: <class 'marshmallow.fields.Integer'>,
<class 'uuid.UUID'>: <class 'marshmallow.fields.UUID'>, <class 'datetime.time'>:
<class 'marshmallow.fields.Time'>, <class 'datetime.date'>: <class
'marshmallow.fields.Date'>, <class 'datetime.timedelta'>: <class
'marshmallow.fields.TimeDelta'>, <class 'decimal.Decimal'>: <class
'marshmallow.fields.Decimal'>}
```

property dict_class: *type*

dump(*obj*: Any, *, *many*: bool | None = None)

Serialize an object to native Python data types according to this *Schema's* fields.

Parameters

- **obj** – The object to serialize.
- **many** – Whether to serialize *obj* as a collection. If *None*, the value for *self.many* is used.

Returns

Serialized data

New in version 1.0.0.

Changed in version 3.0.0b7: This method returns the serialized data rather than a (data, errors) tuple. A `ValidationError` is raised if `obj` is invalid.

Changed in version 3.0.0rc9: Validation no longer occurs upon serialization.

dump(*obj: Any, *args, many: bool | None = None, **kwargs*)

Same as `dump()`, except return a JSON-encoded string.

Parameters

- **obj** – The object to serialize.
- **many** – Whether to serialize *obj* as a collection. If *None*, the value for *self.many* is used.

Returns

A json string

New in version 1.0.0.

Changed in version 3.0.0b7: This method returns the serialized data rather than a (data, errors) tuple. A `ValidationError` is raised if `obj` is invalid.

error_messages: Dict[str, str] = {'unknown': 'Unknown configuration. See https://nitpick.rtf.d.io/en/latest/nitpick_section.html.'}

Overrides for default schema-level error messages

fields: Dict[str, ma_fields.Field]

Dictionary mapping field_names -> Field objects

classmethod from_dict(*fields: dict[str, mallow.fields.Field | type], *, name: str = 'GeneratedSchema') → type*

Generate a *Schema* class given a dictionary of fields.

```
from marshmallow import Schema, fields

PersonSchema = Schema.from_dict({"name": fields.Str()})
print(PersonSchema().load({"name": "David"})) # => {'name': 'David'}
```

Generated schemas are not added to the class registry and therefore cannot be referred to by name in *Nested* fields.

Parameters

- **fields** (*dict*) – Dictionary mapping field names to field instances.
- **name** (*str*) – Optional name for the class, which will appear in the repr for the class.

New in version 3.0.0.

get_attribute(*obj: Any, attr: str, default: Any*)

Defines how to pull values from an object to serialize.

New in version 2.0.0.

Changed in version 3.0.0a1: Changed position of `obj` and `attr`.

handle_error(*error: ValidationError, data: Any, *, many: bool, **kwargs*)

Custom error handler function for the schema.

Parameters

- **error** – The `ValidationError` raised during (de)serialization.

- **data** – The original input data.
- **many** – Value of `many` on dump or load.
- **partial** – Value of `partial` on load.

New in version 2.0.0.

Changed in version 3.0.0rc9: Receives *many* and *partial* (on deserialization) as keyword arguments.

load(*data*: *Mapping*[*str*, *Any*] | *Iterable*[*Mapping*[*str*, *Any*]], *, *many*: *bool* | *None* = *None*, *partial*: *bool* | *Sequence*[*str*] | *AbstractSet*[*str*] | *None* = *None*, *unknown*: *str* | *None* = *None*)

Deserialize a data structure to an object defined by this Schema's fields.

Parameters

- **data** – The data to deserialize.
- **many** – Whether to deserialize *data* as a collection. If *None*, the value for *self.many* is used.
- **partial** – Whether to ignore missing fields and not require any fields declared. Propagates down to `Nested` fields as well. If its value is an iterable, only missing fields listed in that iterable will be ignored. Use dot delimiters to specify nested fields.
- **unknown** – Whether to exclude, include, or raise an error for unknown fields in the data. Use *EXCLUDE*, *INCLUDE* or *RAISE*. If *None*, the value for *self.unknown* is used.

Returns

Serialized data

New in version 1.0.0.

Changed in version 3.0.0b7: This method returns the deserialized data rather than a (data, errors) tuple. A `ValidationError` is raised if invalid data are passed.

loads(*json_data*: *str*, *, *many*: *bool* | *None* = *None*, *partial*: *bool* | *Sequence*[*str*] | *AbstractSet*[*str*] | *None* = *None*, *unknown*: *str* | *None* = *None*, ***kwargs*)

Same as `load()`, except it takes a JSON string as input.

Parameters

- **json_data** – A JSON string of the data to deserialize.
- **many** – Whether to deserialize *obj* as a collection. If *None*, the value for *self.many* is used.
- **partial** – Whether to ignore missing fields and not require any fields declared. Propagates down to `Nested` fields as well. If its value is an iterable, only missing fields listed in that iterable will be ignored. Use dot delimiters to specify nested fields.
- **unknown** – Whether to exclude, include, or raise an error for unknown fields in the data. Use *EXCLUDE*, *INCLUDE* or *RAISE*. If *None*, the value for *self.unknown* is used.

Returns

Serialized data

New in version 1.0.0.

Changed in version 3.0.0b7: This method returns the deserialized data rather than a (data, errors) tuple. A `ValidationError` is raised if invalid data are passed.

on_bind_field(*field_name*: *str*, *field_obj*: `Field`) → *None*

Hook to modify a field when it is bound to the *Schema*.

No-op by default.

opts: `SchemaOpts` = `<marshmallow.schema.SchemaOpts object>`

set_class

alias of `OrderedSet`

validate(*data*: `Mapping[str, Any] | Iterable[Mapping[str, Any]]`, *, *many*: `bool | None = None`, *partial*: `bool | Sequence[str] | AbstractSet[str] | None = None`) → `dict[str, list[str]]`

Validate *data* against the schema, returning a dictionary of validation errors.

Parameters

- **data** – The data to validate.
- **many** – Whether to validate *data* as a collection. If *None*, the value for *self.many* is used.
- **partial** – Whether to ignore missing fields and not require any fields declared. Propagates down to `Nested` fields as well. If its value is an iterable, only missing fields listed in that iterable will be ignored. Use dot delimiters to specify nested fields.

Returns

A dictionary of validation errors.

New in version 1.1.0.

class `nitpick.plugins.json.JsonPlugin`(*info*: `FileInfo`, *expected_config*: `JsonDict`, *autofix*=`False`)

Bases: `NitpickPlugin`

Enforce configurations and autofix JSON files.

Add the configurations for the file name you wish to check. Style example: [the default config for package.json](#).

dirty: `bool`

enforce_rules() → `Iterator[Fuss]`

Enforce rules for missing keys and JSON content.

entry_point() → `Iterator[Fuss]`

Entry point of the Nitpick plugin.

expected_config: `JsonDict`

expected_dict_from_contains_json()

Expected dict created from “contains_json” values.

expected_dict_from_contains_keys()

Expected dict created from “contains_keys” values.

file_path: `Path`

filename = ''

fixable: `bool = True`

Can this plugin modify its files directly? Are the files fixable?

identify_tags: `ClassVar = {'json'}`

Which identify tags this `nitpick.plugins.base.NitpickPlugin` child recognises.

property initial_contents: `str`

Suggest the initial content for this missing file.

property nitpick_file_dict: `Dict[str, Any]`

Nitpick configuration for this file as a TOML dict, taken from the style file.

post_init()

Hook for plugin initialization after the instance was created.

The name mimics `__post_init__()` on dataclasses, without the magic double underscores: [Post-init processing](#)

predefined_special_config() \rightarrow *SpecialConfig*

Create a predefined special configuration for this plugin.

Each plugin can override this method.

report(*violation:* *ViolationEnum*, *blender:* *JsonDict*, *change:* *BaseDoc | None*)

Report a violation while optionally modifying the JSON dict.

skip_empty_suggestion = False

validation_schema

alias of *JsonFileSchema*

violation_base_code: `int = 340`

write_file(*file_exists:* *bool*) \rightarrow *Fuss | None*

Hook to write the new file when autofix mode is on.

Should be used by inherited classes.

write_initial_contents(*doc_class:* *type[nitpick.blender.BaseDoc]*, *expected_dict:* *dict | None = None*)
 \rightarrow `str`

Helper to write initial contents based on a format.

`nitpick.plugins.json.can_handle`(*info:* *FileInfo*) \rightarrow *type[NitpickPlugin] | None*

Handle JSON files.

`nitpick.plugins.json.plugin_class`() \rightarrow *type[nitpick.plugins.base.NitpickPlugin]*

Handle JSON files.

nitpick.plugins.text module

Text files.

class `nitpick.plugins.text.TextItemSchema`(**, only:* *Sequence[str] | AbstractSet[str] | None = None*,
exclude: *Sequence[str] | AbstractSet[str] = ()*, *many:* *bool = False*, *context:* *dict | None = None*, *load_only:* *Sequence[str] | AbstractSet[str] = ()*, *dump_only:* *Sequence[str] | AbstractSet[str] = ()*, *partial:* *bool | Sequence[str] | AbstractSet[str] | None = None*, *unknown:* *str | None = None*)

Bases: *Schema*

Validation schema for the object inside contains.

class `Meta`

Bases: *object*

Options object for a Schema.

Example usage:

```
class Meta:
    fields = ("id", "email", "date_created")
    exclude = ("password", "secret_attribute")
```

Available options:

- **fields:** Tuple or list of fields to include in the serialized result.
- **additional:** Tuple or list of fields to include *in addition to the* explicitly declared fields. `additional` and `fields` are mutually-exclusive options.
- **include:** Dictionary of additional fields to include in the schema. It is usually better to define fields as class variables, but you may need to use this option, e.g., if your fields are Python keywords. May be an *OrderedDict*.
- **exclude:** Tuple or list of fields to exclude in the serialized result. Nested fields can be represented with dot delimiters.
- **dateformat:** Default format for *Date* `<fields.Date>` fields.
- **datetimeformat:** Default format for *DateTime* `<fields.DateTime>` fields.
- **timeformat:** Default format for *Time* `<fields.Time>` fields.
- **render_module:** Module to use for *loads* `<Schema.loads>` and *dumps* `<Schema.dumps>`. Defaults to *json* from the standard library.
- **ordered:** If *True*, output of *Schema.dump* will be a *collections.OrderedDict*.
- **index_errors:** If *True*, errors dictionaries will include the index of invalid items in a collection.
- **load_only:** Tuple or list of fields to exclude from serialized results.
- **dump_only:** Tuple or list of fields to exclude from deserialization
- **unknown:** Whether to exclude, include, or raise an error for unknown fields in the data. Use *EXCLUDE*, *INCLUDE* or *RAISE*.
- **register:** Whether to register the *Schema* with *marshmallow's* internal class registry. Must be *True* if you intend to refer to this *Schema* by class name in *Nested* fields. Only set this to *False* when memory usage is critical. Defaults to *True*.

OPTIONS_CLASS

alias of `SchemaOpts`

```
TYPE_MAPPING: Dict[type, Type[ma_fields.Field]] = {<class 'str'>: <class
'marshmallow.fields.String'>, <class 'bytes'>: <class 'marshmallow.fields.String'>,
<class 'datetime.datetime'>: <class 'marshmallow.fields.DateTime'>, <class
'float'>: <class 'marshmallow.fields.Float'>, <class 'bool'>: <class
'marshmallow.fields.Boolean'>, <class 'tuple'>: <class 'marshmallow.fields.Raw'>,
<class 'list'>: <class 'marshmallow.fields.Raw'>, <class 'set'>: <class
'marshmallow.fields.Raw'>, <class 'int'>: <class 'marshmallow.fields.Integer'>,
<class 'uuid.UUID'>: <class 'marshmallow.fields.UUID'>, <class 'datetime.time'>:
<class 'marshmallow.fields.Time'>, <class 'datetime.date'>: <class
'marshmallow.fields.Date'>, <class 'datetime.timedelta'>: <class
'marshmallow.fields.TimeDelta'>, <class 'decimal.Decimal'>: <class
'marshmallow.fields.Decimal'>}
```

property `dict_class`: `type`

dump(*obj*: Any, *, *many*: bool | None = None)

Serialize an object to native Python data types according to this Schema's fields.

Parameters

- **obj** – The object to serialize.
- **many** – Whether to serialize *obj* as a collection. If *None*, the value for *self.many* is used.

Returns

Serialized data

New in version 1.0.0.

Changed in version 3.0.0b7: This method returns the serialized data rather than a (data, errors) tuple. A `ValidationError` is raised if *obj* is invalid.

Changed in version 3.0.0rc9: Validation no longer occurs upon serialization.

dumps(*obj*: Any, **args*, *many*: bool | None = None, ***kwargs*)

Same as `dump()`, except return a JSON-encoded string.

Parameters

- **obj** – The object to serialize.
- **many** – Whether to serialize *obj* as a collection. If *None*, the value for *self.many* is used.

Returns

A json string

New in version 1.0.0.

Changed in version 3.0.0b7: This method returns the serialized data rather than a (data, errors) tuple. A `ValidationError` is raised if *obj* is invalid.

error_messages: Dict[str, str] = {'unknown': 'Unknown configuration. See <https://nitpick.rtf.d.io/en/latest/plugins.html#text-files>.'}
 Overrides for default schema-level error messages

fields: Dict[str, ma_fields.Field]

Dictionary mapping field_names -> Field objects

classmethod from_dict(*fields*: dict[str, marshmallow.fields.Field | type], *, *name*: str = 'GeneratedSchema') -> type

Generate a *Schema* class given a dictionary of fields.

```
from marshmallow import Schema, fields

PersonSchema = Schema.from_dict({"name": fields.Str()})
print(PersonSchema().load({"name": "David"})) # => {'name': 'David'}
```

Generated schemas are not added to the class registry and therefore cannot be referred to by name in *Nested* fields.

Parameters

- **fields** (*dict*) – Dictionary mapping field names to field instances.
- **name** (*str*) – Optional name for the class, which will appear in the repr for the class.

New in version 3.0.0.

get_attribute(*obj: Any, attr: str, default: Any*)

Defines how to pull values from an object to serialize.

New in version 2.0.0.

Changed in version 3.0.0a1: Changed position of *obj* and *attr*.

handle_error(*error: ValidationError, data: Any, *, many: bool, **kwargs*)

Custom error handler function for the schema.

Parameters

- **error** – The *ValidationError* raised during (de)serialization.
- **data** – The original input data.
- **many** – Value of *many* on dump or load.
- **partial** – Value of *partial* on load.

New in version 2.0.0.

Changed in version 3.0.0rc9: Receives *many* and *partial* (on deserialization) as keyword arguments.

load(*data: Mapping[str, Any] | Iterable[Mapping[str, Any]], *, many: bool | None = None, partial: bool | Sequence[str] | AbstractSet[str] | None = None, unknown: str | None = None*)

Deserialize a data structure to an object defined by this Schema's fields.

Parameters

- **data** – The data to deserialize.
- **many** – Whether to deserialize *data* as a collection. If *None*, the value for *self.many* is used.
- **partial** – Whether to ignore missing fields and not require any fields declared. Propagates down to *Nested* fields as well. If its value is an iterable, only missing fields listed in that iterable will be ignored. Use dot delimiters to specify nested fields.
- **unknown** – Whether to exclude, include, or raise an error for unknown fields in the data. Use *EXCLUDE*, *INCLUDE* or *RAISE*. If *None*, the value for *self.unknown* is used.

Returns

Deserialized data

New in version 1.0.0.

Changed in version 3.0.0b7: This method returns the deserialized data rather than a (*data*, *errors*) tuple. A *ValidationError* is raised if invalid data are passed.

loads(*json_data: str, *, many: bool | None = None, partial: bool | Sequence[str] | AbstractSet[str] | None = None, unknown: str | None = None, **kwargs*)

Same as *load()*, except it takes a JSON string as input.

Parameters

- **json_data** – A JSON string of the data to deserialize.
- **many** – Whether to deserialize *obj* as a collection. If *None*, the value for *self.many* is used.
- **partial** – Whether to ignore missing fields and not require any fields declared. Propagates down to *Nested* fields as well. If its value is an iterable, only missing fields listed in that iterable will be ignored. Use dot delimiters to specify nested fields.
- **unknown** – Whether to exclude, include, or raise an error for unknown fields in the data. Use *EXCLUDE*, *INCLUDE* or *RAISE*. If *None*, the value for *self.unknown* is used.

Returns

Deserialized data

New in version 1.0.0.

Changed in version 3.0.0b7: This method returns the deserialized data rather than a (data, errors) tuple. A `ValidationError` is raised if invalid data are passed.

on_bind_field(*field_name*: str, *field_obj*: Field) → NoneHook to modify a field when it is bound to the *Schema*.

No-op by default.

opts: SchemaOpts = <marshmallow.schema.SchemaOpts object>**set_class**

alias of OrderedSet

validate(*data*: Mapping[str, Any] | Iterable[Mapping[str, Any]], *, *many*: bool | None = None, *partial*: bool | Sequence[str] | AbstractSet[str] | None = None) → dict[str, list[str]]Validate *data* against the schema, returning a dictionary of validation errors.**Parameters**

- **data** – The data to validate.
- **many** – Whether to validate *data* as a collection. If *None*, the value for *self.many* is used.
- **partial** – Whether to ignore missing fields and not require any fields declared. Propagates down to Nested fields as well. If its value is an iterable, only missing fields listed in that iterable will be ignored. Use dot delimiters to specify nested fields.

Returns

A dictionary of validation errors.

New in version 1.1.0.

class nitpick.plugins.text.**TextPlugin**(*info*: FileInfo, *expected_config*: JsonDict, *autofix*=False)Bases: *NitpickPlugin*

Enforce configuration on text files.

To check if some.txt file contains the lines abc and def (in any order):

```

[["some.txt".contains]]
line = "abc"

[["some.txt".contains]]
line = "def"

```

dirty: bool**enforce_rules**() → Iterator[Fuss]

Enforce rules for missing lines.

entry_point() → Iterator[Fuss]

Entry point of the Nitpick plugin.

expected_config: JsonDict**file_path**: Path

filename = ''

fixable: `bool` = `False`

Can this plugin modify its files directly? Are the files fixable?

identify_tags: `ClassVar` = `{'text'}`

Which identify tags this `nitpick.plugins.base.NitpickPlugin` child recognises.

property initial_contents: `str`

Suggest the initial content for this missing file.

property nitpick_file_dict: `Dict[str, Any]`

Nitpick configuration for this file as a TOML dict, taken from the style file.

post_init()

Hook for plugin initialization after the instance was created.

The name mimics `__post_init__()` on dataclasses, without the magic double underscores: [Post-init processing](#)

predefined_special_config() → *SpecialConfig*

Create a predefined special configuration for this plugin.

Each plugin can override this method.

skip_empty_suggestion = `True`

validation_schema

alias of *TextSchema*

violation_base_code: `int` = `350`

write_file(*file_exists*: `bool`) → *Fuss* | `None`

Hook to write the new file when autofix mode is on.

Should be used by inherited classes.

write_initial_contents(*doc_class*: *type[nitpick.blender.BaseDoc]*, *expected_dict*: *dict* | `None` = `None`) → `str`

Helper to write initial contents based on a format.

```
class nitpick.plugins.text.TextSchema(*only: Sequence[str] | AbstractSet[str] | None = None, exclude:  
    Sequence[str] | AbstractSet[str] = (), many: bool = False, context:  
    dict | None = None, load_only: Sequence[str] | AbstractSet[str] =  
    (), dump_only: Sequence[str] | AbstractSet[str] = (), partial: bool |  
    Sequence[str] | AbstractSet[str] | None = None, unknown: str |  
    None = None)
```

Bases: *Schema*

Validation schema for the text file TOML configuration.

class Meta

Bases: `object`

Options object for a Schema.

Example usage:


```
class Meta:
    fields = ("id", "email", "date_created")
    exclude = ("password", "secret_attribute")
```

Available options:

- **fields:** Tuple or list of fields to include in the serialized result.
- **additional:** Tuple or list of fields to include *in addition to the* explicitly declared fields. `additional` and `fields` are mutually-exclusive options.
- **include:** Dictionary of additional fields to include in the schema. It is usually better to define fields as class variables, but you may need to use this option, e.g., if your fields are Python keywords. May be an *OrderedDict*.
- **exclude:** Tuple or list of fields to exclude in the serialized result. Nested fields can be represented with dot delimiters.
- **dateformat:** Default format for *Date* `<fields.Date>` fields.
- **datetimeformat:** Default format for *DateTime* `<fields.DateTime>` fields.
- **timeformat:** Default format for *Time* `<fields.Time>` fields.
- **render_module:** Module to use for *loads* `<Schema.loads>` and *dumps* `<Schema.dumps>`. Defaults to *json* from the standard library.
- **ordered:** If *True*, output of *Schema.dump* will be a *collections.OrderedDict*.
- **index_errors:** If *True*, errors dictionaries will include the index of invalid items in a collection.
- **load_only:** Tuple or list of fields to exclude from serialized results.
- **dump_only:** Tuple or list of fields to exclude from deserialization
- **unknown:** Whether to exclude, include, or raise an error for unknown fields in the data. Use *EXCLUDE*, *INCLUDE* or *RAISE*.
- **register:** Whether to register the *Schema* with *marshmallow's* internal class registry. Must be *True* if you intend to refer to this *Schema* by class name in *Nested* fields. Only set this to *False* when memory usage is critical. Defaults to *True*.

OPTIONS_CLASS

alias of `SchemaOpts`

```
TYPE_MAPPING: Dict[type, Type[ma_fields.Field]] = {<class 'str'>: <class
'marshmallow.fields.String'>, <class 'bytes'>: <class 'marshmallow.fields.String'>,
<class 'datetime.datetime'>: <class 'marshmallow.fields.DateTime'>, <class
'float'>: <class 'marshmallow.fields.Float'>, <class 'bool'>: <class
'marshmallow.fields.Boolean'>, <class 'tuple'>: <class 'marshmallow.fields.Raw'>,
<class 'list'>: <class 'marshmallow.fields.Raw'>, <class 'set'>: <class
'marshmallow.fields.Raw'>, <class 'int'>: <class 'marshmallow.fields.Integer'>,
<class 'uuid.UUID'>: <class 'marshmallow.fields.UUID'>, <class 'datetime.time'>:
<class 'marshmallow.fields.Time'>, <class 'datetime.date'>: <class
'marshmallow.fields.Date'>, <class 'datetime.timedelta'>: <class
'marshmallow.fields.TimeDelta'>, <class 'decimal.Decimal'>: <class
'marshmallow.fields.Decimal'>}
```

property `dict_class`: `type`

dump(*obj*: Any, *, *many*: bool | None = None)

Serialize an object to native Python data types according to this Schema's fields.

Parameters

- **obj** – The object to serialize.
- **many** – Whether to serialize *obj* as a collection. If *None*, the value for *self.many* is used.

Returns

Serialized data

New in version 1.0.0.

Changed in version 3.0.0b7: This method returns the serialized data rather than a (data, errors) tuple. A `ValidationError` is raised if *obj* is invalid.

Changed in version 3.0.0rc9: Validation no longer occurs upon serialization.

dumps(*obj*: Any, **args*, *many*: bool | None = None, ***kwargs*)

Same as `dump()`, except return a JSON-encoded string.

Parameters

- **obj** – The object to serialize.
- **many** – Whether to serialize *obj* as a collection. If *None*, the value for *self.many* is used.

Returns

A json string

New in version 1.0.0.

Changed in version 3.0.0b7: This method returns the serialized data rather than a (data, errors) tuple. A `ValidationError` is raised if *obj* is invalid.

error_messages: Dict[str, str] = {'unknown': 'Unknown configuration. See <https://nitpick.rtf.d.io/en/latest/plugins.html#text-files>.'}
Overrides for default schema-level error messages

fields: Dict[str, ma_fields.Field]

Dictionary mapping field_names -> Field objects

classmethod from_dict(*fields*: dict[str, marshmallow.fields.Field | type], *, *name*: str = 'GeneratedSchema') -> type

Generate a *Schema* class given a dictionary of fields.

```
from marshmallow import Schema, fields

PersonSchema = Schema.from_dict({"name": fields.Str()})
print(PersonSchema().load({"name": "David"})) # => {'name': 'David'}
```

Generated schemas are not added to the class registry and therefore cannot be referred to by name in *Nested* fields.

Parameters

- **fields** (*dict*) – Dictionary mapping field names to field instances.
- **name** (*str*) – Optional name for the class, which will appear in the repr for the class.

New in version 3.0.0.

get_attribute(*obj: Any, attr: str, default: Any*)

Defines how to pull values from an object to serialize.

New in version 2.0.0.

Changed in version 3.0.0a1: Changed position of *obj* and *attr*.

handle_error(*error: ValidationError, data: Any, *, many: bool, **kwargs*)

Custom error handler function for the schema.

Parameters

- **error** – The *ValidationError* raised during (de)serialization.
- **data** – The original input data.
- **many** – Value of *many* on dump or load.
- **partial** – Value of *partial* on load.

New in version 2.0.0.

Changed in version 3.0.0rc9: Receives *many* and *partial* (on deserialization) as keyword arguments.

load(*data: Mapping[str, Any] | Iterable[Mapping[str, Any]], *, many: bool | None = None, partial: bool | Sequence[str] | AbstractSet[str] | None = None, unknown: str | None = None*)

Deserialize a data structure to an object defined by this Schema's fields.

Parameters

- **data** – The data to deserialize.
- **many** – Whether to deserialize *data* as a collection. If *None*, the value for *self.many* is used.
- **partial** – Whether to ignore missing fields and not require any fields declared. Propagates down to *Nested* fields as well. If its value is an iterable, only missing fields listed in that iterable will be ignored. Use dot delimiters to specify nested fields.
- **unknown** – Whether to exclude, include, or raise an error for unknown fields in the data. Use *EXCLUDE*, *INCLUDE* or *RAISE*. If *None*, the value for *self.unknown* is used.

Returns

Deserialized data

New in version 1.0.0.

Changed in version 3.0.0b7: This method returns the deserialized data rather than a (*data*, *errors*) tuple. A *ValidationError* is raised if invalid data are passed.

loads(*json_data: str, *, many: bool | None = None, partial: bool | Sequence[str] | AbstractSet[str] | None = None, unknown: str | None = None, **kwargs*)

Same as *load()*, except it takes a JSON string as input.

Parameters

- **json_data** – A JSON string of the data to deserialize.
- **many** – Whether to deserialize *obj* as a collection. If *None*, the value for *self.many* is used.
- **partial** – Whether to ignore missing fields and not require any fields declared. Propagates down to *Nested* fields as well. If its value is an iterable, only missing fields listed in that iterable will be ignored. Use dot delimiters to specify nested fields.
- **unknown** – Whether to exclude, include, or raise an error for unknown fields in the data. Use *EXCLUDE*, *INCLUDE* or *RAISE*. If *None*, the value for *self.unknown* is used.

Returns

Deserialized data

New in version 1.0.0.

Changed in version 3.0.0b7: This method returns the deserialized data rather than a (data, errors) tuple. A `ValidationError` is raised if invalid data are passed.

on_bind_field(*field_name*: str, *field_obj*: Field) → None

Hook to modify a field when it is bound to the *Schema*.

No-op by default.

opts: SchemaOpts = <marshmallow.schema.SchemaOpts object>

set_class

alias of OrderedSet

validate(*data*: Mapping[str, Any] | Iterable[Mapping[str, Any]], *, *many*: bool | None = None, *partial*: bool | Sequence[str] | AbstractSet[str] | None = None) → dict[str, list[str]]

Validate *data* against the schema, returning a dictionary of validation errors.

Parameters

- **data** – The data to validate.
- **many** – Whether to validate *data* as a collection. If *None*, the value for *self.many* is used.
- **partial** – Whether to ignore missing fields and not require any fields declared. Propagates down to *Nested* fields as well. If its value is an iterable, only missing fields listed in that iterable will be ignored. Use dot delimiters to specify nested fields.

Returns

A dictionary of validation errors.

New in version 1.1.0.

class nitpick.plugins.text.Violations(*value*, *names*=None, *, *module*=None, *qualname*=None, *type*=None, *start*=1, *boundary*=None)

Bases: *ViolationEnum*

Violations for this plugin.

MISSING_LINES = (352, ' has missing lines:')

nitpick.plugins.text.can_handle(*info*: FileInfo) → type[NitpickPlugin] | None

Handle text files.

nitpick.plugins.text.plugin_class() → type[nitpick.plugins.base.NitpickPlugin]

Handle text files.

nitpick.plugins.toml module

TOML files.

class nitpick.plugins.toml.TomlPlugin(*info: FileInfo, expected_config: JsonDict, autofix=False*)

Bases: *NitpickPlugin*

Enforce configurations and autofix TOML files.

E.g.: pyproject.toml (PEP 518).

See also the [tool.poetry] section of the pyproject.toml file.

Style example: Python 3.8 version constraint. There are *many other examples here*.

dirty: bool

enforce_rules() → *Iterator[Fuss]*

Enforce rules for missing key/value pairs in the TOML file.

entry_point() → *Iterator[Fuss]*

Entry point of the Nitpick plugin.

expected_config: JsonDict

file_path: Path

filename = ''

fixable: bool = True

Can this plugin modify its files directly? Are the files fixable?

identify_tags: ClassVar = {'toml'}

Which identify tags this *nitpick.plugins.base.NitpickPlugin* child recognises.

property initial_contents: str

Suggest the initial content for this missing file.

property nitpick_file_dict: Dict[str, Any]

Nitpick configuration for this file as a TOML dict, taken from the style file.

post_init()

Hook for plugin initialization after the instance was created.

The name mimics `__post_init__()` on dataclasses, without the magic double underscores: [Post-init processing](#)

predefined_special_config() → *SpecialConfig*

Create a predefined special configuration for this plugin.

Each plugin can override this method.

report(*violation: ViolationEnum, document: TOMLDocument | None, change: TomlDoc | None, replacement: TomlDoc | None = None*)

Report a violation while optionally modifying the TOML document.

skip_empty_suggestion = False

validation_schema: Schema | None = None

Nested validation field for this file, to be applied in runtime when the validation schema is rebuilt. Useful when you have a strict configuration for a file type (e.g. *nitpick.plugins.json.JsonPlugin*).

violation_base_code: `int = 310`

write_file(*file_exists: bool*) → *Fuss* | *None*

Hook to write the new file when autofix mode is on.

Should be used by inherited classes.

write_initial_contents(*doc_class: type[nitpick.blender.BaseDoc]*, *expected_dict: dict* | *None = None*) → *str*

Helper to write initial contents based on a format.

`nitpick.plugins.toml.can_handle`(*info: FileInfo*) → *type[NitpickPlugin]* | *None*

Handle TOML files.

`nitpick.plugins.toml.plugin_class`() → *type[nitpick.plugins.base.NitpickPlugin]*

Handle TOML files.

nitpick.plugins.yaml module

YAML files.

class `nitpick.plugins.yaml.YamlPlugin`(*info: FileInfo*, *expected_config: JsonDict*, *autofix=False*)

Bases: *NitpickPlugin*

Enforce configurations and autofix YAML files.

- Example: `.pre-commit-config.yaml`.
- Style example: [the default pre-commit hooks](#).

Warning: The plugin tries to preserve comments in the YAML file by using the `ruamel.yaml` package. It works for most cases. If your comment was removed, place them in a different place of the file and try again. If it still doesn't work, please [report a bug](#).

Known issue: lists like `args` and `additional_dependencies` might be joined in a single line, and comments between items will be removed. Move your comments outside these lists, and they should be preserved.

Note: No validation of `.pre-commit-config.yaml` will be done anymore in this generic YAML plugin. Nitpick will not validate hooks and missing keys as it did before; it's not the purpose of this package.

dirty: `bool`

enforce_rules() → *Iterator[Fuss]*

Enforce rules for missing data in the YAML file.

entry_point() → *Iterator[Fuss]*

Entry point of the Nitpick plugin.

expected_config: `JsonDict`

file_path: `Path`

filename = ''

fixable: `bool = True`

Can this plugin modify its files directly? Are the files fixable?

identify_tags: `ClassVar = {'yaml'}`

Which identify tags this `nitpick.plugins.base.NitpickPlugin` child recognises.

property initial_contents: `str`

Suggest the initial content for this missing file.

property nitpick_file_dict: `Dict[str, Any]`

Nitpick configuration for this file as a TOML dict, taken from the style file.

post_init()

Hook for plugin initialization after the instance was created.

The name mimics `__post_init__()` on dataclasses, without the magic double underscores: [Post-init processing](#)

predefined_special_config() \rightarrow `SpecialConfig`

Predefined special config, with list keys for `.pre-commit-config.yaml` and GitHub Workflow files.

report(*violation:* `ViolationEnum`, *yaml_object:* `YamlObject`, *change:* `YamlDoc | None`, *replacement:* `YamlDoc | None = None`)

Report a violation while optionally modifying the YAML document.

skip_empty_suggestion = False

validation_schema: `Schema | None = None`

Nested validation field for this file, to be applied in runtime when the validation schema is rebuilt. Useful when you have a strict configuration for a file type (e.g. `nitpick.plugins.json.JsonPlugin`).

violation_base_code: `int = 360`

write_file(*file_exists:* `bool`) \rightarrow `Fuss | None`

Hook to write the new file when autofix mode is on.

Should be used by inherited classes.

write_initial_contents(*doc_class:* `type[nitpick.blender.BaseDoc]`, *expected_dict:* `dict | None = None`) \rightarrow `str`

Helper to write initial contents based on a format.

`nitpick.plugins.yaml.can_handle`(*info:* `FileInfo`) \rightarrow `type[NitpickPlugin] | None`

Handle YAML files.

`nitpick.plugins.yaml.plugin_class`() \rightarrow `type[nitpick.plugins.base.NitpickPlugin]`

Handle YAML files.

nitpick.resources package

A library of Nitpick styles.

Building blocks that can be combined and reused.

Subpackages

nitpick.resources.any package

Styles for any language.

nitpick.resources.javascript package

Styles for JavaScript.

nitpick.resources.kotlin package

Styles for Kotlin.

nitpick.resources.markdown package

Styles for Markdown files.

nitpick.resources.presets package

Style presets.

nitpick.resources.proto package

Styles for Protobuf files.

nitpick.resources.python package

Styles for Python.

nitpick.resources.shell package

Styles for shell scripts.

nitpick.resources.toml package

Styles for TOML files.

12.1.2 Submodules

nitpick.blender module

Dictionary blender and configuration file formats.

```
class nitpick.blender.BaseDoc(*path: Path | str | None = None, string: str | None = None, obj: Dict[str, Any] | None = None)
```

Bases: `object`

Base class for configuration file formats.

Parameters

- **path** – Path of the config file to be loaded.
- **string** – Config in string format.
- **obj** – Config object (Python dict, YamlDoc, TomlDoc instances).

property as_object: `dict`

String content converted to a Python object (dict, YAML object instance, etc.).

property as_string: `str`

Contents of the file or the original string provided when the instance was created.

abstract load() → `bool`

Load the configuration from a file, a string or a dict.

property reformatted: `str`

Reformat the configuration dict as a new string (it might not match the original string/file contents).

```
class nitpick.blender.Comparison(actual: TBaseDoc, expected: JsonDict, special_config: SpecialConfig)
```

Bases: `object`

A comparison between two dictionaries, computing missing items and differences.

property diff: `TBaseDoc | None`

Different data.

property has_changes: `bool`

Return True if there is a difference or something missing.

property missing: `TBaseDoc | None`

Missing data.

property replace: `TBaseDoc | None`

Data to be replaced.

```
class nitpick.blender.ElementDetail(data: Dict[str, Any] | str | int | float | CommentedMap | List[Any], key: str | list[str], index: int, scalar: bool, compact: str)
```

Bases: `object`

Detailed information about an element of a list.

Method generated by attrs for class ElementDetail.

property cast_to_dict: `Dict[str, Any]`

Data cast to dict, for mypy.

compact: `str`

data: `Dict[str, Any] | str | int | float | CommentedMap | List[Any]`

classmethod from_data(*index: int, data: Dict[str, Any] | str | int | float | CommentedMap | List[Any], jmes_key: str*) → *ElementDetail*

Create an element detail from dict data.

index: `int`

key: `str | list[str]`

scalar: `bool`

class `nitpick.blender.InlineTableTomlDecoder`(*_dict=<class 'dict'>*)

Bases: `TomlDecoder`

A hacky decoder to work around some bug (or unfinished work) in the Python TOML package.

<https://github.com/uiri/toml/issues/362>.

bounded_string(*s*)

embed_comments(*idx, currentlevel*)

get_empty_inline_table()

Hackity hack for a crappy unmaintained package.

Total lack of respect, the guy doesn't even reply: <https://github.com/uiri/toml/issues/361>

get_empty_table()

load_array(*a*)

load_inline_object(*line, currentlevel, multikey=False, multibackslash=False*)

load_line(*line, currentlevel, multikey, multibackslash*)

load_value(*v, strictly_valid=True*)

preserve_comment(*line_no, key, comment, beginline*)

class `nitpick.blender.JsonDoc`(**, path: Path | str | None = None, string: str | None = None, obj: Dict[str, Any] | None = None*)

Bases: `BaseDoc`

JSON configuration format.

property as_object: `dict`

String content converted to a Python object (dict, YAML object instance, etc.).

property as_string: `str`

Contents of the file or the original string provided when the instance was created.

load() → `bool`

Load a JSON file by its path, a string or a dict.

property reformatted: `str`

Reformat the configuration dict as a new string (it might not match the original string/file contents).

class nitpick.blender.**ListDetail**(*data: List[Any] | CommentedSeq, elements: list[nitpick.blender.ElementDetail]*)

Bases: `object`

Detailed info about a list.

Method generated by attrs for class ListDetail.

data: `List[Any] | CommentedSeq`

elements: `list[nitpick.blender.ElementDetail]`

find_by_key(*desired: ElementDetail*) → *ElementDetail | None*

Find an element by key.

classmethod from_data(*data: List[Any] | CommentedSeq, jmes_key: str*) → *ListDetail*

Create a list detail from list data.

nitpick.blender.**SEPARATOR_QUOTED_SPLIT** = '#\$@'

Special unique separator for `nitpick.blender.quoted_split()`.

class nitpick.blender.**SensibleYAML**

Bases: `YAML`

YAML with sensible defaults but an inefficient dump to string.

Output of `dump()` as a string.

typ: 'rt'/None -> RoundTripLoader/RoundTripDumper, (default)

'safe' -> SafeLoader/SafeDumper, 'unsafe' -> normal/unsafe Loader/Dumper 'base' -> baseloader

pure: if True only use Python modules input/output: needed to work as context manager `plug_ins`: a list of plug-in files

Xdump_all(*documents: Any, stream: Any, *, transform: Any = None*) → *Any*

Serialize a sequence of Python objects into a YAML stream.

property block_seq_indent: `Any`

compact(*seq_seq: Any = None, seq_map: Any = None*) → *None*

compose(*stream: Path | Any*) → *Any*

Parse the first YAML document in a stream and produce the corresponding representation tree.

compose_all(*stream: Path | Any*) → *Any*

Parse all YAML documents in a stream and produce corresponding representation trees.

property composer: `Any`

property constructor: `Any`

dump(*data: Path | Any, stream: Any = None, *, transform: Any = None*) → *Any*

dump_all(*documents: Any, stream: Path | Any, *, transform: Any = None*) → *Any*

dumps(*data*) → *str*

Dump to a string... who would want such a thing? One can dump to a file or stdout.

emit(*events: Any, stream: Any*) → *None*

Emit YAML parsing events into a stream. If stream is None, return the produced string instead.

property emitter: *Any*

get_constructor_parser(*stream: Any*) → *Any*

the old cyaml needs special setup, and therefore the stream

get_serializer_representer_emitter(*stream: Any, tica: Any*) → *Any*

property indent: *Any*

load(*stream: Path | Any*) → *Any*

at this point you either have the non-pure Parser (which has its own reader and scanner) or you have the pure Parser. If the pure Parser is set, then set the Reader and Scanner, if not already set. If either the Scanner or Reader are set, you cannot use the non-pure Parser,

so reset it to the pure parser and set the Reader resp. Scanner if necessary

load_all(*stream: Path | Any*) → *Any*

loads(*string: str*)

Load YAML from a string... that unusual use case in a world of files only.

map(***kw: Any*) → *Any*

official_plug_ins() → *Any*

search for list of subdirs that are plug-ins, if `__file__` is not available, e.g. single file installers that are not properly emulating a file-system (issue 324) no plug-ins will be found. If any are packaged, you know which file that are and you can explicitly provide it during instantiation:

```
yaml = ruamel.yaml.YAML(plug_ins=['ruamel/yaml/jinja2/__plug_in__'])
```

parse(*stream: Any*) → *Any*

Parse a YAML stream and produce parsing events.

property parser: *Any*

property reader: *Any*

register_class(*cls: Any*) → *Any*

register a class for dumping/loading - if it has attribute `yaml_tag` use that to register, else use class name - if it has methods `to_yaml/from_yaml` use those to dump/load else dump attributes

as mapping

property representer: *Any*

property resolver: *Any*

scan(*stream: Any*) → *Any*

Scan a YAML stream and produce scanning tokens.

property scanner: *Any*

seq(**args: Any*) → *Any*

serialize(*node: Any, stream: Any | None*) → *Any*

Serialize a representation tree into a YAML stream. If stream is None, return the produced string instead.

serialize_all(*nodes: Any, stream: Any | None*) → *Any*

Serialize a sequence of representation trees into a YAML stream. If stream is None, return the produced string instead.

property serializer: `Any`

property version: `Any | None`

class `nitpick.blender.TomlDoc`(**path*: `Path | str | None = None`, *string*: `str | None = None`, *obj*: `Dict[str, Any] | None = None`, *use_tomlkit*=`False`)

Bases: `BaseDoc`

TOML configuration format.

property as_object: `dict`

String content converted to a Python object (dict, YAML object instance, etc.).

property as_string: `str`

Contents of the file or the original string provided when the instance was created.

load() → `bool`

Load a TOML file by its path, a string or a dict.

property reformatted: `str`

Reformat the configuration dict as a new string (it might not match the original string/file contents).

class `nitpick.blender.YamlDoc`(**path*: `Path | str | None = None`, *string*: `str | None = None`, *obj*: `Dict[str, Any] | None = None`)

Bases: `BaseDoc`

YAML configuration format.

property as_object: `dict`

String content converted to a Python object (dict, YAML object instance, etc.).

property as_string: `str`

Contents of the file or the original string provided when the instance was created.

load() → `bool`

Load a YAML file by its path, a string or a dict.

property reformatted: `str`

Reformat the configuration dict as a new string (it might not match the original string/file contents).

updater: `SensibleYAML`

`nitpick.blender.compare_lists_with_dictdiffer`(*actual*: `list | dict`, *expected*: `list | dict`, *, *return_list*: `bool = True`) → `list | dict`

Compare two lists using dictdiffer.

`nitpick.blender.custom_reducer`(*separator*: `str`) → `Callable`

Custom reducer for `flatten_dict.flatten_dict.flatten()` accepting a separator.

`nitpick.blender.custom_splitter`(*separator*: `str`) → `Callable`

Custom splitter for `flatten_dict.flatten_dict.unflatten()` accepting a separator.

`nitpick.blender.flatten_quotes`(*dict_*: `Dict[str, Any]`, *separator*='\'') → `Dict[str, Any]`

Flatten a dict keeping quotes in keys.

`nitpick.blender.is_scalar`(*value*: `Dict[str, Any] | List[Any] | str | float`) → `bool`

Return True if the value is NOT a dict or a list.

`nitpick.blender.quote_if_dotted(key: str) → str`

Quote the key if it has a dot.

`nitpick.blender.quote_reducer(separator: str) → Callable`

Reducer used to unflatten dicts.

Quote keys when they have dots.

`nitpick.blender.quoted_split(string_: str, separator='.') → list[str]`

Split a string by a separator, but considering quoted parts (single or double quotes).

```
>>> quoted_split("my.key.without.quotes")
['my', 'key', 'without', 'quotes']
>>> quoted_split('"double.quoted.string"')
['double.quoted.string']
>>> quoted_split('"double.quoted.string".and.after')
['double.quoted.string', 'and', 'after']
>>> quoted_split('something.before."double.quoted.string"')
['something', 'before', 'double.quoted.string']
>>> quoted_split("'single.quoted.string'")
['single.quoted.string']
>>> quoted_split("'single.quoted.string'.and.after")
['single.quoted.string', 'and', 'after']
>>> quoted_split("something.before.'single.quoted.string'")
['something', 'before', 'single.quoted.string']
```

`nitpick.blender.quotes_splitter(flat_key: str) → tuple[str, ...]`

Split keys keeping quoted strings together.

`nitpick.blender.replace_or_add_list_element(yaml_obj: CommentedSeq | CommentedMap, element: Any, key: str, index: int) → None`

Replace or add a new element in a YAML sequence of mappings.

`nitpick.blender.search_json(json_data: ElementData, jmespath_expression: ParsedResult | str, default: Any | None = None) → Any`

Search a dictionary or list using a JMESPath expression.

Return a default value if not found.

```
>>> data = {"root": {"app": [1, 2], "test": "something"}}
>>> search_json(data, "root.app", None)
[1, 2]
>>> search_json(data, "root.test", None)
'something'
>>> search_json(data, "root.unknown", "")
''
>>> search_json(data, "root.unknown", None)
```

```
>>> search_json(data, "root.unknown")
```

```
>>> search_json(data, jmespath.compile("root.app"), [])
[1, 2]
>>> search_json(data, jmespath.compile("root.whatever"), "xxx")
'xxx'
>>> search_json(data, "")
```

```
>>> search_json(data, None)
```

Parameters

- **jmespath_expression** – A compiled JMESPath expression or a string with an expression.
- **json_data** – The dictionary to be searched.
- **default** – Default value in case nothing is found.

Returns

The object that was found or the default value.

`nitpick.blender.set_key_if_not_empty(dict_: Dict[str, Any], key: str, value: Any) → None`

Update the dict if the value is valid.

`nitpick.blender.traverse_toml_tree(document: TOMLDocument, dictionary)`

Traverse a TOML document recursively and change values, keeping its formatting and comments.

`nitpick.blender.traverse_yaml_tree(yaml_obj: CommentedSeq | CommentedMap, change: Dict[str, Any])`

Traverse a YAML document recursively and change values, keeping its formatting and comments.

nitpick.cli module

Module that contains the command line app.

Why does this file exist, and why not put this in `__main__`?

You might be tempted to import things from `__main__` later, but that will cause problems: the code will get executed twice:

- **When you run `python -m nitpick` python will execute `__main__.py` as a script.**
That means there won't be any `nitpick.__main__` in `sys.modules`.
- **When you import `__main__` it will get executed again (as a module) because**
there's no `nitpick.__main__` in `sys.modules`.

Also see (1) from <https://click.palletsprojects.com/en/5.x/setuptools/#setuptools-integration>

`nitpick.cli.common_fix_or_check(context, verbose: int, files, check_only: bool) → None`

Common CLI code for both “fix” and “check” commands.

`nitpick.cli.get_nitpick(context: Context) → Nitpick`

Create a Nitpick instance from the click context parameters.

nitpick.compat module

Handle import compatibility issues.

nitpick.config module

Special configurations.

```
class nitpick.config.OverridableConfig(from_plugin: Dict[str, Any] = _Nothing.NOTHING, from_style: Dict[str, Any] = _Nothing.NOTHING, value: Dict[str, Any] = _Nothing.NOTHING)
```

Bases: `object`

Configs formed from defaults from the plugin that can be overridden by the style.

Method generated by attrs for class OverridableConfig.

from_plugin: `Dict[str, Any]`

from_style: `Dict[str, Any]`

value: `Dict[str, Any]`

```
class nitpick.config.SpecialConfig(list_keys: OverridableConfig = _Nothing.NOTHING)
```

Bases: `object`

Special configurations for plugins.

Method generated by attrs for class SpecialConfig.

list_keys: `OverridableConfig`

nitpick.constants module

Constants.

```
class nitpick.constants.CachingEnum(value, names=None, *, module=None, qualname=None, type=None, start=1, boundary=None)
```

Bases: `IntEnum`

Caching modes for styles.

EXPIRES = 3

The cache expires after the configured amount of time (minutes/hours/days).

FOREVER = 2

Once the style(s) are cached, they never expire.

NEVER = 1

Never cache, the style file(s) are always looked-up.

as_integer_ratio()

Return integer ratio.

Return a pair of integers, whose ratio is exactly equal to the original int and with a positive denominator.

```
>>> (10).as_integer_ratio()
(10, 1)
>>> (-10).as_integer_ratio()
(-10, 1)
>>> (0).as_integer_ratio()
(0, 1)
```


bit_count()

Number of ones in the binary representation of the absolute value of self.

Also known as the population count.

```
>>> bin(13)
'0b1101'
>>> (13).bit_count()
3
```

bit_length()

Number of bits necessary to represent self in binary.

```
>>> bin(37)
'0b100101'
>>> (37).bit_length()
6
```

conjugate()

Returns self, the complex conjugate of any int.

denominator

the denominator of a rational number in lowest terms

from_bytes(*byteorder='big', *, signed=False*)

Return the integer represented by the given array of bytes.

bytes

Holds the array of bytes to convert. The argument must either support the buffer protocol or be an iterable object producing bytes. Bytes and bytearray are examples of built-in objects that support the buffer protocol.

byteorder

The byte order used to represent the integer. If *byteorder* is 'big', the most significant byte is at the beginning of the byte array. If *byteorder* is 'little', the most significant byte is at the end of the byte array. To request the native byte order of the host system, use `sys.byteorder` as the byte order value. Default is to use 'big'.

signed

Indicates whether two's complement is used to represent the integer.

imag

the imaginary part of a complex number

numerator

the numerator of a rational number in lowest terms

real

the real part of a complex number

to_bytes(*length=1, byteorder='big', *, signed=False*)

Return an array of bytes representing an integer.

length

Length of bytes object to use. An `OverflowError` is raised if the integer is not representable with the given number of bytes. Default is length 1.

byteorder

The byte order used to represent the integer. If byteorder is 'big', the most significant byte is at the beginning of the byte array. If byteorder is 'little', the most significant byte is at the end of the byte array. To request the native byte order of the host system, use 'sys.byteorder' as the byte order value. Default is to use 'big'.

signed

Determines whether two's complement is used to represent the integer. If signed is False and a negative integer is given, an OverflowError is raised.

```
class nitpick.constants.EmojiEnum(value, names=None, *, module=None, qualname=None, type=None,
                                  start=1, boundary=None)
```

Bases: [Enum](#)

Emojis used in the CLI.

```
CONSTRUCTION = ''
```

```
GREEN_CHECK = ''
```

```
QUESTION_MARK = ''
```

```
SLEEPY_FACE = ''
```

```
STAR_CAKE = ' '
```

```
X_RED_CROSS = ''
```

```
class nitpick.constants.Flake8OptionEnum(value, names=None, *, module=None, qualname=None,
                                          type=None, start=1, boundary=None)
```

Bases: [_OptionMixin](#), [Enum](#)

Options to be used with the flake8 plugin/CLI.

```
OFFLINE = 'Offline mode: no style will be downloaded (no HTTP requests at all)'
```

```
name: str
```

nitpick.core module

The Nitpick application and project-related utilities.

```
class nitpick.core.Configuration(file: Path, doc: TOMLDocument, table: Table, styles: Array,
                                dont_suggest: Array, cache: str)
```

Bases: [object](#)

Configuration read from the [tool.nitpick] table from one of the CONFIG_FILES.

```
cache: str
```

```
doc: TOMLDocument
```

```
dont_suggest: Array
```

```
file: Path
```

```
styles: Array
```

```
table: Table
```

class nitpick.core.NitpickBases: `object`

The Nitpick API.

configured_files(**partial_names*: *str*) → `list[pathlib.Path]`

List of files configured in the Nitpick style.

Filter only the selected partial names.

echo(*message*: *str*)

Echo a message on the terminal, with the relative path at the beginning.

enforce_present_absent(**partial_names*: *str*) → `Iterator[Fuss]`

Enforce files that should be present or absent.

Parameters**partial_names** – Names of the files to enforce configs for.**Returns**

Fuss generator.

enforce_style(**partial_names*: *str*, *autofix*=*True*) → `Iterator[Fuss]`

Read the merged style and enforce the rules in it.

1. Get all root keys from the merged style (every key is a filename, except “nitpick”).
2. For each file name, find the plugin(s) that can handle the file.

Parameters

- **partial_names** – Names of the files to enforce configs for.
- **autofix** – Flag to modify files, if the plugin supports it (default: True).

Returns

Fuss generator.

init(*project_root*: *PathOrStr* | *None* = *None*, *offline*: *bool* | *None* = *None*) → `Nitpick`

Initialize attributes of the singleton.

offline: `bool`**project**: `Project`**run**(**partial_names*: *str*, *autofix*=*False*) → `Iterator[Fuss]`

Run Nitpick.

Parameters

- **partial_names** – Names of the files to enforce configs for.
- **autofix** – Flag to modify files, if the plugin supports it (default: True).

Returns

Fuss generator.

classmethod singleton() → `Nitpick`

Return a single instance of the class.

```
class nitpick.core.Project(root: PathOrStr | None = None)
```

Bases: `object`

A project to be nitpicked.

```
config_file() → Path | None
```

Determine which config file to use.

```
config_file_or_default() → Path
```

Return a config file if found, or the default one.

```
merge_styles(offline: bool) → Iterator[Fuss]
```

Merge one or multiple style files.

```
property plugin_manager: PluginManager
```

Load all defined plugins.

```
read_configuration() → Configuration
```

Return the `[tool.nitpick]` table from the configuration file.

Optionally, validate it against a Marshmallow schema.

```
property root: Path
```

Root dir of the project.

```
suggest_styles(library_path_str: PathOrStr | None) → list[str]
```

Suggest styles based on the files in the project root (skipping Git ignored files).

```
class nitpick.core.ToolNitpickSectionSchema(*, only: Sequence[str] | AbstractSet[str] | None = None,
                                             exclude: Sequence[str] | AbstractSet[str] = (), many: bool
                                             = False, context: dict | None = None, load_only:
                                             Sequence[str] | AbstractSet[str] = (), dump_only:
                                             Sequence[str] | AbstractSet[str] = (), partial: bool |
                                             Sequence[str] | AbstractSet[str] | None = None, unknown:
                                             str | None = None)
```

Bases: `BaseNitpickSchema`

Validation schema for the `[tool.nitpick]` table on `pyproject.toml`.

```
class Meta
```

Bases: `object`

Options object for a Schema.

Example usage:

```
class Meta:
    fields = ("id", "email", "date_created")
    exclude = ("password", "secret_attribute")
```

Available options:

- **fields:** Tuple or list of fields to include in the serialized result.
- **additional:** Tuple or list of fields to include *in addition to the* explicitly declared fields. `additional` and `fields` are mutually-exclusive options.
- **include:** Dictionary of additional fields to include in the schema. It is usually better to define fields as class variables, but you may need to use this option, e.g., if your fields are Python keywords. May be an `OrderedDict`.

- **exclude:** Tuple or list of fields to exclude in the serialized result.
Nested fields can be represented with dot delimiters.
- **dateformat:** Default format for *Date* <*fields.Date*> fields.
- **datetimeformat:** Default format for *DateTime* <*fields.DateTime*> fields.
- **timeformat:** Default format for *Time* <*fields.Time*> fields.
- **render_module:** Module to use for *loads* <*Schema.loads*> and *dumps* <*Schema.dumps*>.
Defaults to *json* from the standard library.
- **ordered:** If *True*, output of *Schema.dump* will be a *collections.OrderedDict*.
- **index_errors:** If *True*, errors dictionaries will include the index of invalid items in a collection.
- **load_only:** Tuple or list of fields to exclude from serialized results.
- **dump_only:** Tuple or list of fields to exclude from deserialization
- **unknown:** Whether to exclude, include, or raise an error for unknown fields in the data. Use *EXCLUDE*, *INCLUDE* or *RAISE*.
- **register:** Whether to register the *Schema* with *marshmallow's* internal class registry. Must be *True* if you intend to refer to this *Schema* by class name in *Nested* fields. Only set this to *False* when memory usage is critical. Defaults to *True*.

OPTIONS_CLASS

alias of `SchemaOpts`

```
TYPE_MAPPING: Dict[type, Type[ma_fields.Field]] = {<class 'str'>: <class
'marshmallow.fields.String'>, <class 'bytes'>: <class 'marshmallow.fields.String'>,
<class 'datetime.datetime'>: <class 'marshmallow.fields.DateTime'>, <class
'float'>: <class 'marshmallow.fields.Float'>, <class 'bool'>: <class
'marshmallow.fields.Boolean'>, <class 'tuple'>: <class 'marshmallow.fields.Raw'>,
<class 'list'>: <class 'marshmallow.fields.Raw'>, <class 'set'>: <class
'marshmallow.fields.Raw'>, <class 'int'>: <class 'marshmallow.fields.Integer'>,
<class 'uuid.UUID'>: <class 'marshmallow.fields.UUID'>, <class 'datetime.time'>:
<class 'marshmallow.fields.Time'>, <class 'datetime.date'>: <class
'marshmallow.fields.Date'>, <class 'datetime.timedelta'>: <class
'marshmallow.fields.TimeDelta'>, <class 'decimal.Decimal'>: <class
'marshmallow.fields.Decimal'>}
```

property `dict_class`: `type`

`dump(obj: Any, *, many: bool | None = None)`

Serialize an object to native Python data types according to this Schema's fields.

Parameters

- **obj** – The object to serialize.
- **many** – Whether to serialize *obj* as a collection. If *None*, the value for *self.many* is used.

Returns

Serialized data

New in version 1.0.0.

Changed in version 3.0.0b7: This method returns the serialized data rather than a (data, errors) tuple. A `ValidationError` is raised if *obj* is invalid.

Changed in version 3.0.0rc9: Validation no longer occurs upon serialization.

dump(*obj*: Any, **args*, *many*: bool | None = None, ***kwargs*)

Same as `dump()`, except return a JSON-encoded string.

Parameters

- **obj** – The object to serialize.
- **many** – Whether to serialize *obj* as a collection. If *None*, the value for *self.many* is used.

Returns

A json string

New in version 1.0.0.

Changed in version 3.0.0b7: This method returns the serialized data rather than a (data, errors) tuple. A `ValidationError` is raised if *obj* is invalid.

error_messages: Dict[str, str] = {'unknown': 'Unknown configuration. See <https://nitpick.rtfid.io/en/latest/configuration.html>.'}
 Overrides for default schema-level error messages

fields: Dict[str, ma_fields.Field]

Dictionary mapping field_names -> Field objects

classmethod from_dict(*fields*: dict[str, marshmallow.fields.Field | type], *, *name*: str = 'GeneratedSchema') → type

Generate a *Schema* class given a dictionary of fields.

```
from marshmallow import Schema, fields

PersonSchema = Schema.from_dict({"name": fields.Str()})
print(PersonSchema().load({"name": "David"})) # => {'name': 'David'}
```

Generated schemas are not added to the class registry and therefore cannot be referred to by name in *Nested* fields.

Parameters

- **fields** (*dict*) – Dictionary mapping field names to field instances.
- **name** (*str*) – Optional name for the class, which will appear in the repr for the class.

New in version 3.0.0.

get_attribute(*obj*: Any, *attr*: str, *default*: Any)

Defines how to pull values from an object to serialize.

New in version 2.0.0.

Changed in version 3.0.0a1: Changed position of *obj* and *attr*.

handle_error(*error*: ValidationError, *data*: Any, *, *many*: bool, ***kwargs*)

Custom error handler function for the schema.

Parameters

- **error** – The `ValidationError` raised during (de)serialization.
- **data** – The original input data.
- **many** – Value of *many* on dump or load.
- **partial** – Value of *partial* on load.

New in version 2.0.0.

Changed in version 3.0.0rc9: Receives *many* and *partial* (on deserialization) as keyword arguments.

load(*data*: *Mapping*[*str*, *Any*] | *Iterable*[*Mapping*[*str*, *Any*]], *, *many*: *bool* | *None* = *None*, *partial*: *bool* | *Sequence*[*str*] | *AbstractSet*[*str*] | *None* = *None*, *unknown*: *str* | *None* = *None*)

Deserialize a data structure to an object defined by this Schema's fields.

Parameters

- **data** – The data to deserialize.
- **many** – Whether to deserialize *data* as a collection. If *None*, the value for *self.many* is used.
- **partial** – Whether to ignore missing fields and not require any fields declared. Propagates down to *Nested* fields as well. If its value is an iterable, only missing fields listed in that iterable will be ignored. Use dot delimiters to specify nested fields.
- **unknown** – Whether to exclude, include, or raise an error for unknown fields in the data. Use *EXCLUDE*, *INCLUDE* or *RAISE*. If *None*, the value for *self.unknown* is used.

Returns

Deserialized data

New in version 1.0.0.

Changed in version 3.0.0b7: This method returns the deserialized data rather than a (data, errors) tuple. A *ValidationError* is raised if invalid data are passed.

loads(*json_data*: *str*, *, *many*: *bool* | *None* = *None*, *partial*: *bool* | *Sequence*[*str*] | *AbstractSet*[*str*] | *None* = *None*, *unknown*: *str* | *None* = *None*, ***kwargs*)

Same as *load()*, except it takes a JSON string as input.

Parameters

- **json_data** – A JSON string of the data to deserialize.
- **many** – Whether to deserialize *obj* as a collection. If *None*, the value for *self.many* is used.
- **partial** – Whether to ignore missing fields and not require any fields declared. Propagates down to *Nested* fields as well. If its value is an iterable, only missing fields listed in that iterable will be ignored. Use dot delimiters to specify nested fields.
- **unknown** – Whether to exclude, include, or raise an error for unknown fields in the data. Use *EXCLUDE*, *INCLUDE* or *RAISE*. If *None*, the value for *self.unknown* is used.

Returns

Deserialized data

New in version 1.0.0.

Changed in version 3.0.0b7: This method returns the deserialized data rather than a (data, errors) tuple. A *ValidationError* is raised if invalid data are passed.

on_bind_field(*field_name*: *str*, *field_obj*: *Field*) → *None*

Hook to modify a field when it is bound to the *Schema*.

No-op by default.

opts: *SchemaOpts* = <marshmallow.schema.SchemaOpts object>

set_class

alias of *OrderedSet*

validate(*data*: *Mapping*[*str*, *Any*] | *Iterable*[*Mapping*[*str*, *Any*]], *, *many*: *bool* | *None* = *None*, *partial*: *bool* | *Sequence*[*str*] | *AbstractSet*[*str*] | *None* = *None*) → *dict*[*str*, *list*[*str*]]

Validate *data* against the schema, returning a dictionary of validation errors.

Parameters

- **data** – The data to validate.
- **many** – Whether to validate *data* as a collection. If *None*, the value for *self.many* is used.
- **partial** – Whether to ignore missing fields and not require any fields declared. Propagates down to *Nested* fields as well. If its value is an iterable, only missing fields listed in that iterable will be ignored. Use dot delimiters to specify nested fields.

Returns

A dictionary of validation errors.

New in version 1.1.0.

`nitpick.core.confirm_project_root`(*dir_*: *PathOrStr* | *None* = *None*) → *Path*

Confirm this is the root dir of the project (the one that has one of the `ROOT_FILES`).

`nitpick.core.find_main_python_file`(*root_dir*: *Path*) → *Path*

Find the main Python file in the root dir, the one that will be used to report Flake8 warnings.

The search order is: 1. Python files that belong to the root dir of the project (e.g.: `setup.py`, `autoapp.py`). 2. `manage.py`: they can be on the root or on a subdir (Django projects). 3. Any other `*.py` Python file on the root dir and subdir. This avoid long recursions when there is a `node_modules` subdir for instance.

nitpick.exceptions module

Nitpick exceptions.

class `nitpick.exceptions.Deprecation`

Bases: `object`

All deprecation messages in a single class.

When it's time to break compatibility, remove a method/warning below, and older config files will trigger validation errors on Nitpick.

static `jsonfile_section`(*style_errors*: *dict*[*str*, *Any*]) → *bool*

The `[nitpick.JSONFile]` is not needed anymore; JSON files are now detected by the extension.

static `pre_commit_repos_with_yaml_key`() → *bool*

The pre-commit config should not have the “`repos.yaml`” key anymore; this is the old style.

Slight breaking change in the TOML config format: ditching the old TOML config.

static `pre_commit_without_dash`(*path_from_root*: *str*) → *bool*

The pre-commit config should start with a dot on the config file.

exception `nitpick.exceptions.QuitComplainingError`(*violations*: *Fuss* | *list*[*Fuss*])

Bases: `Exception`

Quit complaining and exit the application.

add_note()

`Exception.add_note(note)` – add a note to the exception

args

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

nitpick.exceptions.**pretty_exception**(err: *Exception*, message: *str* = "")

Return a pretty error message with the full path of the Exception.

nitpick.fields module

Custom Marshmallow fields and validators.

class nitpick.fields.**Dict**(keys: *Field* | *type* | *None* = *None*, values: *Field* | *type* | *None* = *None*, ****kwargs**)

Bases: *Mapping*

A dict field. Supports dicts and dict-like objects. Extends Mapping with dict as the mapping_type.

Example:

```
numbers = fields.Dict(keys=fields.Str(), values=fields.Float())
```

Parameters

kwargs – The same keyword arguments that Mapping receives.

New in version 2.1.0.

property context

The context dictionary for the parent Schema.

property default

default_error_messages = {'invalid': 'Not a valid mapping type.'}

Default error messages.

deserialize(value: *Any*, attr: *str* | *None* = *None*, data: *Mapping*[*str*, *Any*] | *None* = *None*, ****kwargs**)

Deserialize value.

Parameters

- **value** – The value to deserialize.
- **attr** – The attribute/key in *data* to deserialize.
- **data** – The raw input data passed to *Schema.load*.
- **kwargs** – Field-specific keyword arguments.

Raises

ValidationError – If an invalid value is passed or if a required value is missing.

fail(key: *str*, ****kwargs**)

Helper method that raises a *ValidationError* with an error message from self.error_messages.

Deprecated since version 3.0.0: Use *make_error* <marshmallow.fields.Field.make_error> instead.

get_value(obj, attr, accessor=*None*, default=<marshmallow.missing>)

Return the value for a given key from an object.

Parameters

- **obj** (*object*) – The object to get the value from.

- **attr** (*str*) – The attribute/key in *obj* to get the value from.
- **accessor** (*callable*) – A callable used to retrieve the value of *attr* from the object *obj*. Defaults to *marshmallow.utils.get_value*.

make_error(*key: str, **kwargs*) → *ValidationError*

Helper method to make a *ValidationError* with an error message from *self.error_messages*.

mapping_type

alias of *dict*

property missing

name = *None*

parent = *None*

root = *None*

serialize(*attr: str, obj: Any, accessor: Callable[[Any, str, Any], Any] | None = None, **kwargs*)

Pulls the value for the given key from the object, applies the field's formatting and returns the result.

Parameters

- **attr** – The attribute/key to get from the object.
- **obj** – The object to access the attribute/key from.
- **accessor** – Function used to access values from *obj*.
- **kwargs** – Field-specific keyword arguments.

```
class nitpick.fields.Field(*load_default: ~typing.Any = <marshmallow.missing>, missing: ~typing.Any = <marshmallow.missing>, dump_default: ~typing.Any = <marshmallow.missing>, default: ~typing.Any = <marshmallow.missing>, data_key: str | None = None, attribute: str | None = None, validate: None | ~typing.Callable[[~typing.Any], ~typing.Any] | ~typing.Iterable[~typing.Callable[[~typing.Any], ~typing.Any]] = None, required: bool = False, allow_none: bool | None = None, load_only: bool = False, dump_only: bool = False, error_messages: dict[str, str] | None = None, metadata: ~typing.Mapping[str, ~typing.Any] | None = None, **additional_metadata)
```

Bases: *FieldABC*

Basic field from which other fields should extend. It applies no formatting by default, and should only be used in cases where data does not need to be formatted before being serialized or deserialized. On error, the name of the field will be returned.

Parameters

- **dump_default** – If set, this value will be used during serialization if the input value is missing. If not set, the field will be excluded from the serialized output if the input value is missing. May be a value or a callable.
- **load_default** – Default deserialization value for the field if the field is not found in the input data. May be a value or a callable.
- **data_key** – The name of the dict key in the external representation, i.e. the input of *load* and the output of *dump*. If *None*, the key will match the name of the field.

- **attribute** – The name of the attribute to get the value from when serializing. If *None*, assumes the attribute has the same name as the field. Note: This should only be used for very specific use cases such as outputting multiple fields for a single attribute. In most cases, you should use `data_key` instead.
- **validate** – Validator or collection of validators that are called during deserialization. Validator takes a field’s input value as its only parameter and returns a boolean. If it returns *False*, an `ValidationError` is raised.
- **required** – Raise a `ValidationError` if the field value is not supplied during deserialization.
- **allow_none** – Set this to *True* if *None* should be considered a valid value during validation/deserialization. If `load_default=None` and `allow_none` is unset, will default to *True*. Otherwise, the default is *False*.
- **load_only** – If *True* skip this field during serialization, otherwise its value will be present in the serialized data.
- **dump_only** – If *True* skip this field during deserialization, otherwise its value will be present in the deserialized object. In the context of an HTTP API, this effectively marks the field as “read-only”.
- **error_messages** (*dict*) – Overrides for `Field.default_error_messages`.
- **metadata** – Extra information to be stored as field metadata.

Changed in version 2.0.0: Removed `error` parameter. Use `error_messages` instead.

Changed in version 2.0.0: Added `allow_none` parameter, which makes validation/deserialization of *None* consistent across fields.

Changed in version 2.0.0: Added `load_only` and `dump_only` parameters, which allow field skipping during the (de)serialization process.

Changed in version 2.0.0: Added `missing` parameter, which indicates the value for a field if the field is not found during deserialization.

Changed in version 2.0.0: `default` value is only used if explicitly set. Otherwise, missing values inputs are excluded from serialized output.

Changed in version 3.0.0b8: Add `data_key` parameter for the specifying the key in the input and output data. This parameter replaced both `load_from` and `dump_to`.

property context

The context dictionary for the parent Schema.

property default

```
default_error_messages = {'null': 'Field may not be null.', 'required': 'Missing data for required field.', 'validator_failed': 'Invalid value.'}
```

Default error messages for various kinds of errors. The keys in this dictionary are passed to `Field.make_error`. The values are error messages passed to `marshmallow.exceptions.ValidationError`.

```
deserialize(value: Any, attr: str | None = None, data: Mapping[str, Any] | None = None, **kwargs)
```

Deserialize value.

Parameters

- **value** – The value to deserialize.
- **attr** – The attribute/key in `data` to deserialize.

- **data** – The raw input data passed to *Schema.load*.
- **kwargs** – Field-specific keyword arguments.

Raises

ValidationError – If an invalid value is passed or if a required value is missing.

fail(*key: str, **kwargs*)

Helper method that raises a *ValidationError* with an error message from `self.error_messages`.

Deprecated since version 3.0.0: Use `make_error <marshmallow.fields.Field.make_error>` instead.

get_value(*obj, attr, accessor=None, default=<marshmallow.missing>*)

Return the value for a given key from an object.

Parameters

- **obj** (*object*) – The object to get the value from.
- **attr** (*str*) – The attribute/key in *obj* to get the value from.
- **accessor** (*callable*) – A callable used to retrieve the value of *attr* from the object *obj*. Defaults to `marshmallow.utils.get_value`.

make_error(*key: str, **kwargs*) → *ValidationError*

Helper method to make a *ValidationError* with an error message from `self.error_messages`.

property missing

name = None

parent = None

root = None

serialize(*attr: str, obj: Any, accessor: Callable[[Any, str, Any], Any] | None = None, **kwargs*)

Pulls the value for the given key from the object, applies the field's formatting and returns the result.

Parameters

- **attr** – The attribute/key to get from the object.
- **obj** – The object to access the attribute/key from.
- **accessor** – Function used to access values from *obj*.
- **kwargs** – Field-specific keyword arguments.

class nitpick.fields.**List**(*cls_or_instance: Field | type, **kwargs*)

Bases: *Field*

A list field, composed with another *Field* class or instance.

Example:

```
numbers = fields.List(fields.Float())
```

Parameters

- **cls_or_instance** – A field class or instance.
- **kwargs** – The same keyword arguments that *Field* receives.

Changed in version 2.0.0: The `allow_none` parameter now applies to deserialization and has the same semantics as the other fields.

Changed in version 3.0.0rc9: Does not serialize scalar values to single-item lists.

property context

The context dictionary for the parent Schema.

property default

default_error_messages = {'invalid': 'Not a valid list.'}

Default error messages.

deserialize(*value: Any, attr: str | None = None, data: Mapping[str, Any] | None = None, **kwargs*)

Deserialize value.

Parameters

- **value** – The value to deserialize.
- **attr** – The attribute/key in *data* to deserialize.
- **data** – The raw input data passed to *Schema.load*.
- **kwargs** – Field-specific keyword arguments.

Raises

ValidationError – If an invalid value is passed or if a required value is missing.

fail(*key: str, **kwargs*)

Helper method that raises a *ValidationError* with an error message from `self.error_messages`.

Deprecated since version 3.0.0: Use `make_error <marshmallow.fields.Field.make_error>` instead.

get_value(*obj, attr, accessor=None, default=<marshmallow.missing>*)

Return the value for a given key from an object.

Parameters

- **obj** (*object*) – The object to get the value from.
- **attr** (*str*) – The attribute/key in *obj* to get the value from.
- **accessor** (*callable*) – A callable used to retrieve the value of *attr* from the object *obj*. Defaults to `marshmallow.utils.get_value`.

make_error(*key: str, **kwargs*) → *ValidationError*

Helper method to make a *ValidationError* with an error message from `self.error_messages`.

property missing

name = None

parent = None

root = None

serialize(*attr: str, obj: Any, accessor: Callable[[Any, str, Any], Any] | None = None, **kwargs*)

Pulls the value for the given key from the object, applies the field's formatting and returns the result.

Parameters

- **attr** – The attribute/key to get from the object.
- **obj** – The object to access the attribute/key from.

- **accessor** – Function used to access values from obj.
- **kwargs** – Field-specific keyword arguments.

```
class nitpick.fields.Nested(nested: ~marshmallow.base.SchemaABC | type | str | dict[str,
marshmallow.fields.Field | type] | ~typing.Callable[[],
~marshmallow.base.SchemaABC | dict[str, marshmallow.fields.Field | type]], *,
dump_default: ~typing.Any = <marshmallow.missing>, default: ~typing.Any =
<marshmallow.missing>, only: ~typing.Sequence[str] | ~typing.AbstractSet[str]
| None = None, exclude: ~typing.Sequence[str] | ~typing.AbstractSet[str] = (),
many: bool = False, unknown: str | None = None, **kwargs)
```

Bases: *Field*

Allows you to nest a *Schema* inside a field.

Examples:

```
class ChildSchema(Schema):
    id = fields.Str()
    name = fields.Str()
    # Use lambda functions when you need two-way nesting or self-nesting
    parent = fields.Nested(lambda: ParentSchema(only=("id",)), dump_only=True)
    siblings = fields.List(fields.Nested(lambda: ChildSchema(only=("id", "name"))))

class ParentSchema(Schema):
    id = fields.Str()
    children = fields.List(
        fields.Nested(ChildSchema(only=("id", "parent", "siblings")))
    )
    spouse = fields.Nested(lambda: ParentSchema(only=("id",)))
```

When passing a *Schema* <marshmallow.Schema> instance as the first argument, the instance's *exclude*, *only*, and *many* attributes will be respected.

Therefore, when passing the *exclude*, *only*, or *many* arguments to *fields.Nested*, you should pass a *Schema* <marshmallow.Schema> class (not an instance) as the first argument.

```
# Yes
author = fields.Nested(UserSchema, only=('id', 'name'))

# No
author = fields.Nested(UserSchema(), only=('id', 'name'))
```

Parameters

- **nested** – *Schema* instance, class, class name (string), dictionary, or callable that returns a *Schema* or dictionary. Dictionaries are converted with *Schema.from_dict*.
- **exclude** – A list or tuple of fields to exclude.
- **only** – A list or tuple of fields to marshal. If *None*, all fields are marshalled. This parameter takes precedence over *exclude*.
- **many** – Whether the field is a collection of objects.
- **unknown** – Whether to exclude, include, or raise an error for unknown fields in the data. Use *EXCLUDE*, *INCLUDE* or *RAISE*.
- **kwargs** – The same keyword arguments that *Field* receives.

property context

The context dictionary for the parent Schema.

property default

default_error_messages = {'type': 'Invalid type.'}

Default error messages.

deserialize(*value: Any, attr: str | None = None, data: Mapping[str, Any] | None = None, **kwargs*)

Deserialize value.

Parameters

- **value** – The value to deserialize.
- **attr** – The attribute/key in *data* to deserialize.
- **data** – The raw input data passed to *Schema.load*.
- **kwargs** – Field-specific keyword arguments.

Raises

ValidationError – If an invalid value is passed or if a required value is missing.

fail(*key: str, **kwargs*)

Helper method that raises a *ValidationError* with an error message from *self.error_messages*.

Deprecated since version 3.0.0: Use *make_error* <*marshmallow.fields.Field.make_error*> instead.

get_value(*obj, attr, accessor=None, default=<marshmallow.missing>*)

Return the value for a given key from an object.

Parameters

- **obj** (*object*) – The object to get the value from.
- **attr** (*str*) – The attribute/key in *obj* to get the value from.
- **accessor** (*callable*) – A callable used to retrieve the value of *attr* from the object *obj*. Defaults to *marshmallow.utils.get_value*.

make_error(*key: str, **kwargs*) → *ValidationError*

Helper method to make a *ValidationError* with an error message from *self.error_messages*.

property missing

name = None

parent = None

root = None

property schema

The nested Schema object.

Changed in version 1.0.0: Renamed from *serializer* to *schema*.

serialize(*attr: str, obj: Any, accessor: Callable[[Any, str, Any], Any] | None = None, **kwargs*)

Pulls the value for the given key from the object, applies the field's formatting and returns the result.

Parameters

- **attr** – The attribute/key to get from the object.

- **obj** – The object to access the attribute/key from.
- **accessor** – Function used to access values from obj.
- **kwargs** – Field-specific keyword arguments.

```
class nitpick.fields.String(*, load_default: ~typing.Any = <marshmallow.missing>, missing: ~typing.Any = <marshmallow.missing>, dump_default: ~typing.Any = <marshmallow.missing>, default: ~typing.Any = <marshmallow.missing>, data_key: str | None = None, attribute: str | None = None, validate: None | ~typing.Callable[~typing.Any], ~typing.Any] | ~typing.Iterable[~typing.Callable[~typing.Any], ~typing.Any]] = None, required: bool = False, allow_none: bool | None = None, load_only: bool = False, dump_only: bool = False, error_messages: dict[str, str] | None = None, metadata: ~typing.Mapping[str, ~typing.Any] | None = None, **additional_metadata)
```

Bases: *Field*

A string field.

Parameters

kwargs – The same keyword arguments that *Field* receives.

property context

The context dictionary for the parent Schema.

property default

```
default_error_messages = {'invalid': 'Not a valid string.', 'invalid_utf8': 'Not a valid utf-8 string.'}
```

Default error messages.

```
deserialize(value: Any, attr: str | None = None, data: Mapping[str, Any] | None = None, **kwargs)
```

Deserialize value.

Parameters

- **value** – The value to deserialize.
- **attr** – The attribute/key in *data* to deserialize.
- **data** – The raw input data passed to *Schema.load*.
- **kwargs** – Field-specific keyword arguments.

Raises

ValidationError – If an invalid value is passed or if a required value is missing.

```
fail(key: str, **kwargs)
```

Helper method that raises a *ValidationError* with an error message from *self.error_messages*.

Deprecated since version 3.0.0: Use *make_error* `<marshmallow.fields.Field.make_error>` instead.

```
get_value(obj, attr, accessor=None, default=<marshmallow.missing>)
```

Return the value for a given key from an object.

Parameters

- **obj** (*object*) – The object to get the value from.
- **attr** (*str*) – The attribute/key in *obj* to get the value from.

- **accessor** (*callable*) – A callable used to retrieve the value of *attr* from the object *obj*. Defaults to *marshmallow.utils.get_value*.

make_error(*key: str, **kwargs*) → *ValidationError*

Helper method to make a *ValidationError* with an error message from *self.error_messages*.

property missing

name = *None*

parent = *None*

root = *None*

serialize(*attr: str, obj: Any, accessor: Callable[[Any, str, Any], Any] | None = None, **kwargs*)

Pulls the value for the given key from the object, applies the field's formatting and returns the result.

Parameters

- **attr** – The attribute/key to get from the object.
- **obj** – The object to access the attribute/key from.
- **accessor** – Function used to access values from *obj*.
- **kwargs** – Field-specific keyword arguments.

`nitpick.fields.URL`

alias of `Url`

nitpick.flake8 module

Flake8 plugin to check files.

class `nitpick.flake8.NitpickFlake8Extension`(*tree=None, filename='(none)'*)

Bases: `object`

Main class for the flake8 extension.

Method generated by attrs for class `NitpickFlake8Extension`.

static `add_options`(*option_manager: OptionManager*)

Add the offline option.

build_flake8_error(*obj: Fuss*) → `Tuple[int, int, str, Type]`

Return a flake8 error from a fuss.

collect_errors() → `Iterator[Fuss]`

Collect all possible Nitpick errors.

name = `'nitpick'`

static `parse_options`(*option_manager: OptionManager, options, args*)

Create the Nitpick app, set logging from the verbose flags, set offline mode.

This function is called only once by flake8, so it's a good place to create the app.

run() → `Iterator[Tuple[int, int, str, Type]]`

Run the check plugin.

version = `'0.35.0'`

nitpick.generic module

Generic functions and classes.

`nitpick.generic.filter_names(iterable: Iterable, *partial_names: str) → list[str]`

Filter names and keep only the desired partial names.

Exclude the project name automatically.

```
>>> file_list = ['requirements.txt', 'tox.ini', 'setup.py', 'nitpick']
>>> filter_names(file_list)
['requirements.txt', 'tox.ini', 'setup.py']
>>> filter_names(file_list, 'ini', '.py')
['tox.ini', 'setup.py']
```

```
>>> mapping = {'requirements.txt': None, 'tox.ini': 1, 'setup.py': 2, 'nitpick': 3}
>>> filter_names(mapping)
['requirements.txt', 'tox.ini', 'setup.py']
>>> filter_names(file_list, 'x')
['requirements.txt', 'tox.ini']
```

`nitpick.generic.get_global_gitignore_path() → Path | None`

Get the path to the global Git ignore file.

`nitpick.generic.glob_files(dir_: Path, file_patterns: Iterable[str]) → set[pathlib.Path]`

Search a directory looking for file patterns.

`nitpick.generic.glob_non_ignored_files(root_dir: Path, pattern: str = '**/*') → Iterable[Path]`

Glob all files in the root dir that are not ignored by Git.

`nitpick.generic.relative_to_current_dir(path_or_str: PathOrStr | None) → str`

Return a relative path to the current dir or an absolute path.

`nitpick.generic.url_to_python_path(url: furl) → Path`

Convert the segments of a file URL to a path.

nitpick.schemas module

Marshmallow schemas.

```
class nitpick.schemas.BaseNitpickSchema(*, only: Sequence[str] | AbstractSet[str] | None = None,
                                         exclude: Sequence[str] | AbstractSet[str] = (), many: bool =
                                         False, context: dict | None = None, load_only: Sequence[str] |
                                         AbstractSet[str] = (), dump_only: Sequence[str] |
                                         AbstractSet[str] = (), partial: bool | Sequence[str] |
                                         AbstractSet[str] | None = None, unknown: str | None = None)
```

Bases: `Schema`

Base schema for all others, with default error messages.

class `Meta`

Bases: `object`

Options object for a Schema.

Example usage:

```
class Meta:
    fields = ("id", "email", "date_created")
    exclude = ("password", "secret_attribute")
```

Available options:

- **fields:** Tuple or list of fields to include in the serialized result.
- **additional:** Tuple or list of fields to include *in addition to the* explicitly declared fields. `additional` and `fields` are mutually-exclusive options.
- **include:** Dictionary of additional fields to include in the schema. It is usually better to define fields as class variables, but you may need to use this option, e.g., if your fields are Python keywords. May be an *OrderedDict*.
- **exclude:** Tuple or list of fields to exclude in the serialized result. Nested fields can be represented with dot delimiters.
- **dateformat:** Default format for *Date* `<fields.Date>` fields.
- **datetimeformat:** Default format for *DateTime* `<fields.DateTime>` fields.
- **timeformat:** Default format for *Time* `<fields.Time>` fields.
- **render_module:** Module to use for *loads* `<Schema.loads>` and *dumps* `<Schema.dumps>`. Defaults to *json* from the standard library.
- **ordered:** If *True*, output of *Schema.dump* will be a *collections.OrderedDict*.
- **index_errors:** If *True*, errors dictionaries will include the index of invalid items in a collection.
- **load_only:** Tuple or list of fields to exclude from serialized results.
- **dump_only:** Tuple or list of fields to exclude from deserialization
- **unknown:** Whether to exclude, include, or raise an error for unknown fields in the data. Use *EXCLUDE*, *INCLUDE* or *RAISE*.
- **register:** Whether to register the *Schema* with *marshmallow's* internal class registry. Must be *True* if you intend to refer to this *Schema* by class name in *Nested* fields. Only set this to *False* when memory usage is critical. Defaults to *True*.

OPTIONS_CLASS

alias of `SchemaOpts`

```
TYPE_MAPPING: Dict[type, Type[ma_fields.Field]] = {<class 'str'>: <class
'marshmallow.fields.String'>, <class 'bytes'>: <class 'marshmallow.fields.String'>,
<class 'datetime.datetime'>: <class 'marshmallow.fields.DateTime'>, <class
'float'>: <class 'marshmallow.fields.Float'>, <class 'bool'>: <class
'marshmallow.fields.Boolean'>, <class 'tuple'>: <class 'marshmallow.fields.Raw'>,
<class 'list'>: <class 'marshmallow.fields.Raw'>, <class 'set'>: <class
'marshmallow.fields.Raw'>, <class 'int'>: <class 'marshmallow.fields.Integer'>,
<class 'uuid.UUID'>: <class 'marshmallow.fields.UUID'>, <class 'datetime.time'>:
<class 'marshmallow.fields.Time'>, <class 'datetime.date'>: <class
'marshmallow.fields.Date'>, <class 'datetime.timedelta'>: <class
'marshmallow.fields.TimeDelta'>, <class 'decimal.Decimal'>: <class
'marshmallow.fields.Decimal'>}
```

property `dict_class`: `type`

dump(*obj*: Any, *, *many*: bool | None = None)

Serialize an object to native Python data types according to this Schema's fields.

Parameters

- **obj** – The object to serialize.
- **many** – Whether to serialize *obj* as a collection. If *None*, the value for *self.many* is used.

Returns

Serialized data

New in version 1.0.0.

Changed in version 3.0.0b7: This method returns the serialized data rather than a (data, errors) tuple. A `ValidationError` is raised if *obj* is invalid.

Changed in version 3.0.0rc9: Validation no longer occurs upon serialization.

dumps(*obj*: Any, **args*, *many*: bool | None = None, ***kwargs*)

Same as `dump()`, except return a JSON-encoded string.

Parameters

- **obj** – The object to serialize.
- **many** – Whether to serialize *obj* as a collection. If *None*, the value for *self.many* is used.

Returns

A json string

New in version 1.0.0.

Changed in version 3.0.0b7: This method returns the serialized data rather than a (data, errors) tuple. A `ValidationError` is raised if *obj* is invalid.

error_messages: Dict[str, str] = {'unknown': 'Unknown configuration. See https://nitpick.rtf.d.io/en/latest/nitpick_section.html.'}
 Overrides for default schema-level error messages

fields: Dict[str, ma_fields.Field]

Dictionary mapping field_names -> Field objects

classmethod from_dict(*fields*: dict[str, marshmallow.fields.Field | type], *, *name*: str = 'GeneratedSchema') -> type

Generate a *Schema* class given a dictionary of fields.

```
from marshmallow import Schema, fields

PersonSchema = Schema.from_dict({"name": fields.Str()})
print(PersonSchema().load({"name": "David"})) # => {'name': 'David'}
```

Generated schemas are not added to the class registry and therefore cannot be referred to by name in *Nested* fields.

Parameters

- **fields** (*dict*) – Dictionary mapping field names to field instances.
- **name** (*str*) – Optional name for the class, which will appear in the repr for the class.

New in version 3.0.0.

get_attribute(*obj: Any, attr: str, default: Any*)

Defines how to pull values from an object to serialize.

New in version 2.0.0.

Changed in version 3.0.0a1: Changed position of *obj* and *attr*.

handle_error(*error: ValidationError, data: Any, *, many: bool, **kwargs*)

Custom error handler function for the schema.

Parameters

- **error** – The *ValidationError* raised during (de)serialization.
- **data** – The original input data.
- **many** – Value of *many* on dump or load.
- **partial** – Value of *partial* on load.

New in version 2.0.0.

Changed in version 3.0.0rc9: Receives *many* and *partial* (on deserialization) as keyword arguments.

load(*data: Mapping[str, Any] | Iterable[Mapping[str, Any]], *, many: bool | None = None, partial: bool | Sequence[str] | AbstractSet[str] | None = None, unknown: str | None = None*)

Deserialize a data structure to an object defined by this Schema's fields.

Parameters

- **data** – The data to deserialize.
- **many** – Whether to deserialize *data* as a collection. If *None*, the value for *self.many* is used.
- **partial** – Whether to ignore missing fields and not require any fields declared. Propagates down to *Nested* fields as well. If its value is an iterable, only missing fields listed in that iterable will be ignored. Use dot delimiters to specify nested fields.
- **unknown** – Whether to exclude, include, or raise an error for unknown fields in the data. Use *EXCLUDE*, *INCLUDE* or *RAISE*. If *None*, the value for *self.unknown* is used.

Returns

Deserialized data

New in version 1.0.0.

Changed in version 3.0.0b7: This method returns the deserialized data rather than a (*data*, *errors*) tuple. A *ValidationError* is raised if invalid data are passed.

loads(*json_data: str, *, many: bool | None = None, partial: bool | Sequence[str] | AbstractSet[str] | None = None, unknown: str | None = None, **kwargs*)

Same as *load()*, except it takes a JSON string as input.

Parameters

- **json_data** – A JSON string of the data to deserialize.
- **many** – Whether to deserialize *obj* as a collection. If *None*, the value for *self.many* is used.
- **partial** – Whether to ignore missing fields and not require any fields declared. Propagates down to *Nested* fields as well. If its value is an iterable, only missing fields listed in that iterable will be ignored. Use dot delimiters to specify nested fields.
- **unknown** – Whether to exclude, include, or raise an error for unknown fields in the data. Use *EXCLUDE*, *INCLUDE* or *RAISE*. If *None*, the value for *self.unknown* is used.

Returns

Deserialized data

New in version 1.0.0.

Changed in version 3.0.0b7: This method returns the deserialized data rather than a (data, errors) tuple. A `ValidationError` is raised if invalid data are passed.

on_bind_field(*field_name*: str, *field_obj*: Field) → None

Hook to modify a field when it is bound to the *Schema*.

No-op by default.

opts: `SchemaOpts` = <marshmallow.schema.SchemaOpts object>

set_class

alias of `OrderedSet`

validate(*data*: Mapping[str, Any] | Iterable[Mapping[str, Any]], *, *many*: bool | None = None, *partial*: bool | Sequence[str] | AbstractSet[str] | None = None) → dict[str, list[str]]

Validate *data* against the schema, returning a dictionary of validation errors.

Parameters

- **data** – The data to validate.
- **many** – Whether to validate *data* as a collection. If *None*, the value for *self.many* is used.
- **partial** – Whether to ignore missing fields and not require any fields declared. Propagates down to `Nested` fields as well. If its value is an iterable, only missing fields listed in that iterable will be ignored. Use dot delimiters to specify nested fields.

Returns

A dictionary of validation errors.

New in version 1.1.0.

```
class nitpick.schemas.BaseStyleSchema(*, only: Sequence[str] | AbstractSet[str] | None = None, exclude: Sequence[str] | AbstractSet[str] = (), many: bool = False, context: dict | None = None, load_only: Sequence[str] | AbstractSet[str] = (), dump_only: Sequence[str] | AbstractSet[str] = (), partial: bool | Sequence[str] | AbstractSet[str] | None = None, unknown: str | None = None)
```

Bases: `Schema`

Base validation schema for style files.

Dynamic fields will be added to it later.

class Meta

Bases: `object`

Options object for a `Schema`.

Example usage:

```
class Meta:
    fields = ("id", "email", "date_created")
    exclude = ("password", "secret_attribute")
```

Available options:

- **fields**: Tuple or list of fields to include in the serialized result.

- **additional:** Tuple or list of fields to include *in addition* to the explicitly declared fields. `additional` and `fields` are mutually-exclusive options.
- **include:** Dictionary of additional fields to include in the schema. It is usually better to define fields as class variables, but you may need to use this option, e.g., if your fields are Python keywords. May be an *OrderedDict*.
- **exclude:** Tuple or list of fields to exclude in the serialized result. Nested fields can be represented with dot delimiters.
- `dateformat`: Default format for *Date* `<fields.Date>` fields.
- `datetimeformat`: Default format for *DateTime* `<fields.DateTime>` fields.
- `timeformat`: Default format for *Time* `<fields.Time>` fields.
- **render_module:** Module to use for *loads* `<Schema.loads>` and *dumps* `<Schema.dumps>`. Defaults to *json* from the standard library.
- `ordered`: If *True*, output of *Schema.dump* will be a *collections.OrderedDict*.
- **index_errors:** If *True*, errors dictionaries will include the index of invalid items in a collection.
- `load_only`: Tuple or list of fields to exclude from serialized results.
- `dump_only`: Tuple or list of fields to exclude from deserialization
- **unknown:** Whether to exclude, include, or raise an error for unknown fields in the data. Use *EXCLUDE*, *INCLUDE* or *RAISE*.
- **register:** Whether to register the *Schema* with *marshmallow's* internal class registry. Must be *True* if you intend to refer to this *Schema* by class name in *Nested* fields. Only set this to *False* when memory usage is critical. Defaults to *True*.

OPTIONS_CLASS

alias of *SchemaOpts*

```
TYPE_MAPPING: Dict[type, Type[ma_fields.Field]] = {<class 'str'>: <class
'marshmallow.fields.String'>, <class 'bytes'>: <class 'marshmallow.fields.String'>,
<class 'datetime.datetime'>: <class 'marshmallow.fields.DateTime'>, <class
'float'>: <class 'marshmallow.fields.Float'>, <class 'bool'>: <class
'marshmallow.fields.Boolean'>, <class 'tuple'>: <class 'marshmallow.fields.Raw'>,
<class 'list'>: <class 'marshmallow.fields.Raw'>, <class 'set'>: <class
'marshmallow.fields.Raw'>, <class 'int'>: <class 'marshmallow.fields.Integer'>,
<class 'uuid.UUID'>: <class 'marshmallow.fields.UUID'>, <class 'datetime.time'>:
<class 'marshmallow.fields.Time'>, <class 'datetime.date'>: <class
'marshmallow.fields.Date'>, <class 'datetime.timedelta'>: <class
'marshmallow.fields.TimeDelta'>, <class 'decimal.Decimal'>: <class
'marshmallow.fields.Decimal'>}
```

property `dict_class`: `type`

`dump(obj: Any, *, many: bool | None = None)`

Serialize an object to native Python data types according to this *Schema's* fields.

Parameters

- `obj` – The object to serialize.
- `many` – Whether to serialize *obj* as a collection. If *None*, the value for *self.many* is used.

Returns

Serialized data

New in version 1.0.0.

Changed in version 3.0.0b7: This method returns the serialized data rather than a (data, errors) tuple. A `ValidationError` is raised if `obj` is invalid.

Changed in version 3.0.0rc9: Validation no longer occurs upon serialization.

dumps(*obj*: Any, *args, many: bool | None = None, **kwargs)Same as `dump()`, except return a JSON-encoded string.**Parameters**

- **obj** – The object to serialize.
- **many** – Whether to serialize *obj* as a collection. If `None`, the value for `self.many` is used.

Returns

A json string

New in version 1.0.0.

Changed in version 3.0.0b7: This method returns the serialized data rather than a (data, errors) tuple. A `ValidationError` is raised if `obj` is invalid.

error_messages: Dict[str, str] = {'unknown': 'Unknown file. See <https://nitpick.rtfid.io/en/latest/plugins.html>.'}
Overrides for default schema-level error messages**fields**: Dict[str, ma_fields.Field]

Dictionary mapping field_names -> Field objects

classmethod from_dict(fields: dict[str, marshmallow.fields.Field | type], *, name: str = 'GeneratedSchema') -> typeGenerate a *Schema* class given a dictionary of fields.

```
from marshmallow import Schema, fields

PersonSchema = Schema.from_dict({"name": fields.Str()})
print(PersonSchema().load({"name": "David"})) # => {'name': 'David'}
```

Generated schemas are not added to the class registry and therefore cannot be referred to by name in *Nested* fields.

Parameters

- **fields** (*dict*) – Dictionary mapping field names to field instances.
- **name** (*str*) – Optional name for the class, which will appear in the `repr` for the class.

New in version 3.0.0.

get_attribute(*obj*: Any, attr: str, default: Any)

Defines how to pull values from an object to serialize.

New in version 2.0.0.

Changed in version 3.0.0a1: Changed position of `obj` and `attr`.

handle_error(*error*: *ValidationError*, *data*: *Any*, *, *many*: *bool*, ***kwargs*)

Custom error handler function for the schema.

Parameters

- **error** – The *ValidationError* raised during (de)serialization.
- **data** – The original input data.
- **many** – Value of *many* on dump or load.
- **partial** – Value of *partial* on load.

New in version 2.0.0.

Changed in version 3.0.0rc9: Receives *many* and *partial* (on deserialization) as keyword arguments.

load(*data*: *Mapping*[*str*, *Any*] | *Iterable*[*Mapping*[*str*, *Any*]], *, *many*: *bool* | *None* = *None*, *partial*: *bool* | *Sequence*[*str*] | *AbstractSet*[*str*] | *None* = *None*, *unknown*: *str* | *None* = *None*)

Deserialize a data structure to an object defined by this Schema's fields.

Parameters

- **data** – The data to deserialize.
- **many** – Whether to deserialize *data* as a collection. If *None*, the value for *self.many* is used.
- **partial** – Whether to ignore missing fields and not require any fields declared. Propagates down to *Nested* fields as well. If its value is an iterable, only missing fields listed in that iterable will be ignored. Use dot delimiters to specify nested fields.
- **unknown** – Whether to exclude, include, or raise an error for unknown fields in the data. Use *EXCLUDE*, *INCLUDE* or *RAISE*. If *None*, the value for *self.unknown* is used.

Returns

Deserialized data

New in version 1.0.0.

Changed in version 3.0.0b7: This method returns the deserialized data rather than a (*data*, *errors*) tuple. A *ValidationError* is raised if invalid data are passed.

loads(*json_data*: *str*, *, *many*: *bool* | *None* = *None*, *partial*: *bool* | *Sequence*[*str*] | *AbstractSet*[*str*] | *None* = *None*, *unknown*: *str* | *None* = *None*, ***kwargs*)

Same as *load()*, except it takes a JSON string as input.

Parameters

- **json_data** – A JSON string of the data to deserialize.
- **many** – Whether to deserialize *obj* as a collection. If *None*, the value for *self.many* is used.
- **partial** – Whether to ignore missing fields and not require any fields declared. Propagates down to *Nested* fields as well. If its value is an iterable, only missing fields listed in that iterable will be ignored. Use dot delimiters to specify nested fields.
- **unknown** – Whether to exclude, include, or raise an error for unknown fields in the data. Use *EXCLUDE*, *INCLUDE* or *RAISE*. If *None*, the value for *self.unknown* is used.

Returns

Deserialized data

New in version 1.0.0.

Changed in version 3.0.0b7: This method returns the deserialized data rather than a (*data*, *errors*) tuple. A *ValidationError* is raised if invalid data are passed.

on_bind_field(*field_name*: str, *field_obj*: Field) → None

Hook to modify a field when it is bound to the *Schema*.

No-op by default.

opts: SchemaOpts = <marshmallow.schema.SchemaOpts object>

set_class

alias of OrderedSet

validate(*data*: Mapping[str, Any] | Iterable[Mapping[str, Any]], *, *many*: bool | None = None, *partial*: bool | Sequence[str] | AbstractSet[str] | None = None) → dict[str, list[str]]

Validate *data* against the schema, returning a dictionary of validation errors.

Parameters

- **data** – The data to validate.
- **many** – Whether to validate *data* as a collection. If *None*, the value for *self.many* is used.
- **partial** – Whether to ignore missing fields and not require any fields declared. Propagates down to *Nested* fields as well. If its value is an iterable, only missing fields listed in that iterable will be ignored. Use dot delimiters to specify nested fields.

Returns

A dictionary of validation errors.

New in version 1.1.0.

```
class nitpick.schemas.IniSchema(*, only: Sequence[str] | AbstractSet[str] | None = None, exclude: Sequence[str] | AbstractSet[str] = (), many: bool = False, context: dict | None = None, load_only: Sequence[str] | AbstractSet[str] = (), dump_only: Sequence[str] | AbstractSet[str] = (), partial: bool | Sequence[str] | AbstractSet[str] | None = None, unknown: str | None = None)
```

Bases: *BaseNitpickSchema*

Validation schema for INI files.

class Meta

Bases: *object*

Options object for a Schema.

Example usage:

```
class Meta:
    fields = ("id", "email", "date_created")
    exclude = ("password", "secret_attribute")
```

Available options:

- **fields**: Tuple or list of fields to include in the serialized result.
- **additional**: **Tuple or list of fields to include in addition to the** explicitly declared fields. *additional* and *fields* are mutually-exclusive options.
- **include**: **Dictionary of additional fields to include in the schema. It is** usually better to define fields as class variables, but you may need to use this option, e.g., if your fields are Python keywords. May be an *OrderedDict*.
- **exclude**: **Tuple or list of fields to exclude in the serialized result.** Nested fields can be represented with dot delimiters.

- `dateformat`: Default format for *Date* <*fields.Date*> fields.
- `datetimeformat`: Default format for *DateTime* <*fields.DateTime*> fields.
- `timeformat`: Default format for *Time* <*fields.Time*> fields.
- **`render_module`**: Module to use for *loads* <*Schema.loads*> and *dumps* <*Schema.dumps*>. Defaults to *json* from the standard library.
- `ordered`: If *True*, output of *Schema.dump* will be a *collections.OrderedDict*.
- **`index_errors`**: If *True*, errors dictionaries will include the index of invalid items in a collection.
- `load_only`: Tuple or list of fields to exclude from serialized results.
- `dump_only`: Tuple or list of fields to exclude from deserialization
- **`unknown`**: Whether to exclude, include, or raise an error for unknown fields in the data. Use *EXCLUDE*, *INCLUDE* or *RAISE*.
- **`register`**: Whether to register the *Schema* with *marshmallow*'s internal class registry. Must be *True* if you intend to refer to this *Schema* by class name in *Nested* fields. Only set this to *False* when memory usage is critical. Defaults to *True*.

OPTIONS_CLASS

alias of `SchemaOpts`

```
TYPE_MAPPING: Dict[type, Type[ma_fields.Field]] = {<class 'str'>: <class
'marshmallow.fields.String'>, <class 'bytes'>: <class 'marshmallow.fields.String'>,
<class 'datetime.datetime'>: <class 'marshmallow.fields.DateTime'>, <class
'float'>: <class 'marshmallow.fields.Float'>, <class 'bool'>: <class
'marshmallow.fields.Boolean'>, <class 'tuple'>: <class 'marshmallow.fields.Raw'>,
<class 'list'>: <class 'marshmallow.fields.Raw'>, <class 'set'>: <class
'marshmallow.fields.Raw'>, <class 'int'>: <class 'marshmallow.fields.Integer'>,
<class 'uuid.UUID'>: <class 'marshmallow.fields.UUID'>, <class 'datetime.time'>:
<class 'marshmallow.fields.Time'>, <class 'datetime.date'>: <class
'marshmallow.fields.Date'>, <class 'datetime.timedelta'>: <class
'marshmallow.fields.TimeDelta'>, <class 'decimal.Decimal'>: <class
'marshmallow.fields.Decimal'>}
```

property `dict_class`: `type`

`dump(obj: Any, *, many: bool | None = None)`

Serialize an object to native Python data types according to this *Schema*'s fields.

Parameters

- `obj` – The object to serialize.
- `many` – Whether to serialize *obj* as a collection. If *None*, the value for *self.many* is used.

Returns

Serialized data

New in version 1.0.0.

Changed in version 3.0.0b7: This method returns the serialized data rather than a `(data, errors)` tuple. A `ValidationError` is raised if *obj* is invalid.

Changed in version 3.0.0rc9: Validation no longer occurs upon serialization.

dump_fields: `Dict[str, ma_fields.Field]`

dump(*obj: Any, *args, many: bool | None = None, **kwargs*)

Same as `dump()`, except return a JSON-encoded string.

Parameters

- **obj** – The object to serialize.
- **many** – Whether to serialize *obj* as a collection. If *None*, the value for *self.many* is used.

Returns

A json string

New in version 1.0.0.

Changed in version 3.0.0b7: This method returns the serialized data rather than a (data, errors) tuple. A `ValidationError` is raised if *obj* is invalid.

error_messages: `Dict[str, str] = {'unknown': 'Unknown configuration. See https://nitpick.rtfid.io/en/latest/nitpick_section.html#comma-separated-values.'}`

Overrides for default schema-level error messages

exclude: `set[Any] | MutableSet[Any]`

fields: `Dict[str, ma_fields.Field]`

Dictionary mapping field_names -> Field objects

classmethod from_dict(*fields: dict[str, marshmallow.fields.Field | type], *, name: str = 'GeneratedSchema') → type*

Generate a *Schema* class given a dictionary of fields.

```
from marshmallow import Schema, fields

PersonSchema = Schema.from_dict({"name": fields.Str()})
print(PersonSchema().load({"name": "David"})) # => {'name': 'David'}
```

Generated schemas are not added to the class registry and therefore cannot be referred to by name in *Nested* fields.

Parameters

- **fields** (*dict*) – Dictionary mapping field names to field instances.
- **name** (*str*) – Optional name for the class, which will appear in the repr for the class.

New in version 3.0.0.

get_attribute(*obj: Any, attr: str, default: Any*)

Defines how to pull values from an object to serialize.

New in version 2.0.0.

Changed in version 3.0.0a1: Changed position of *obj* and *attr*.

handle_error(*error: ValidationError, data: Any, *, many: bool, **kwargs*)

Custom error handler function for the schema.

Parameters

- **error** – The `ValidationError` raised during (de)serialization.
- **data** – The original input data.

- **many** – Value of `many` on dump or load.
- **partial** – Value of `partial` on load.

New in version 2.0.0.

Changed in version 3.0.0rc9: Receives *many* and *partial* (on deserialization) as keyword arguments.

load(*data*: *Mapping*[*str*, *Any*] | *Iterable*[*Mapping*[*str*, *Any*]], *, *many*: *bool* | *None* = *None*, *partial*: *bool* | *Sequence*[*str*] | *AbstractSet*[*str*] | *None* = *None*, *unknown*: *str* | *None* = *None*)

Deserialize a data structure to an object defined by this Schema's fields.

Parameters

- **data** – The data to deserialize.
- **many** – Whether to deserialize *data* as a collection. If *None*, the value for *self.many* is used.
- **partial** – Whether to ignore missing fields and not require any fields declared. Propagates down to *Nested* fields as well. If its value is an iterable, only missing fields listed in that iterable will be ignored. Use dot delimiters to specify nested fields.
- **unknown** – Whether to exclude, include, or raise an error for unknown fields in the data. Use *EXCLUDE*, *INCLUDE* or *RAISE*. If *None*, the value for *self.unknown* is used.

Returns

Deserialized data

New in version 1.0.0.

Changed in version 3.0.0b7: This method returns the deserialized data rather than a (data, errors) tuple. A `ValidationError` is raised if invalid data are passed.

load_fields: `Dict[str, ma_fields.Field]`

loads(*json_data*: *str*, *, *many*: *bool* | *None* = *None*, *partial*: *bool* | *Sequence*[*str*] | *AbstractSet*[*str*] | *None* = *None*, *unknown*: *str* | *None* = *None*, **kwargs)

Same as `load()`, except it takes a JSON string as input.

Parameters

- **json_data** – A JSON string of the data to deserialize.
- **many** – Whether to deserialize *obj* as a collection. If *None*, the value for *self.many* is used.
- **partial** – Whether to ignore missing fields and not require any fields declared. Propagates down to *Nested* fields as well. If its value is an iterable, only missing fields listed in that iterable will be ignored. Use dot delimiters to specify nested fields.
- **unknown** – Whether to exclude, include, or raise an error for unknown fields in the data. Use *EXCLUDE*, *INCLUDE* or *RAISE*. If *None*, the value for *self.unknown* is used.

Returns

Deserialized data

New in version 1.0.0.

Changed in version 3.0.0b7: This method returns the deserialized data rather than a (data, errors) tuple. A `ValidationError` is raised if invalid data are passed.

on_bind_field(*field_name*: *str*, *field_obj*: `Field`) → *None*

Hook to modify a field when it is bound to the *Schema*.

No-op by default.

`opts: SchemaOpts = <marshmallow.schema.SchemaOpts object>`

`set_class`

alias of `OrderedSet`

`validate`(*data: Mapping[str, Any] | Iterable[Mapping[str, Any]], *, many: bool | None = None, partial: bool | Sequence[str] | AbstractSet[str] | None = None*) → dict[str, list[str]]

Validate *data* against the schema, returning a dictionary of validation errors.

Parameters

- **data** – The data to validate.
- **many** – Whether to validate *data* as a collection. If *None*, the value for *self.many* is used.
- **partial** – Whether to ignore missing fields and not require any fields declared. Propagates down to `Nested` fields as well. If its value is an iterable, only missing fields listed in that iterable will be ignored. Use dot delimiters to specify nested fields.

Returns

A dictionary of validation errors.

New in version 1.1.0.

```
class nitpick.schemas.NitpickFilesSectionSchema(*, only: Sequence[str] | AbstractSet[str] | None = None, exclude: Sequence[str] | AbstractSet[str] = (), many: bool = False, context: dict | None = None, load_only: Sequence[str] | AbstractSet[str] = (), dump_only: Sequence[str] | AbstractSet[str] = (), partial: bool | Sequence[str] | AbstractSet[str] | None = None, unknown: str | None = None)
```

Bases: `BaseNitpickSchema`

Validation schema for the `[nitpick.files]` section on the style file.

`class Meta`

Bases: `object`

Options object for a Schema.

Example usage:

```
class Meta:
    fields = ("id", "email", "date_created")
    exclude = ("password", "secret_attribute")
```

Available options:

- **fields**: Tuple or list of fields to include in the serialized result.
- **additional**: Tuple or list of fields to include *in addition to the* explicitly declared fields. `additional` and `fields` are mutually-exclusive options.
- **include**: Dictionary of additional fields to include in the schema. It is usually better to define fields as class variables, but you may need to use this option, e.g., if your fields are Python keywords. May be an `OrderedDict`.
- **exclude**: Tuple or list of fields to exclude in the serialized result. Nested fields can be represented with dot delimiters.
- **dateformat**: Default format for `Date <fields.Date>` fields.

- `datetimeformat`: Default format for *DateTime* `<fields.DateTime>` fields.
- `timeformat`: Default format for *Time* `<fields.Time>` fields.
- **`render_module`**: Module to use for *loads* `<Schema.loads>` and *dumps* `<Schema.dumps>`. Defaults to *json* from the standard library.
- `ordered`: If *True*, output of *Schema.dump* will be a *collections.OrderedDict*.
- **`index_errors`**: If *True*, errors dictionaries will include the index of invalid items in a collection.
- `load_only`: Tuple or list of fields to exclude from serialized results.
- `dump_only`: Tuple or list of fields to exclude from deserialization
- **`unknown`**: Whether to exclude, include, or raise an error for unknown fields in the data. Use *EXCLUDE*, *INCLUDE* or *RAISE*.
- **`register`**: Whether to register the *Schema* with *marshmallow's* internal class registry. Must be *True* if you intend to refer to this *Schema* by class name in *Nested* fields. Only set this to *False* when memory usage is critical. Defaults to *True*.

OPTIONS_CLASS

alias of *SchemaOpts*

```
TYPE_MAPPING: Dict[type, Type[ma_fields.Field]] = {<class 'str'>: <class
'marshmallow.fields.String'>, <class 'bytes'>: <class 'marshmallow.fields.String'>,
<class 'datetime.datetime'>: <class 'marshmallow.fields.DateTime'>, <class
'float'>: <class 'marshmallow.fields.Float'>, <class 'bool'>: <class
'marshmallow.fields.Boolean'>, <class 'tuple'>: <class 'marshmallow.fields.Raw'>,
<class 'list'>: <class 'marshmallow.fields.Raw'>, <class 'set'>: <class
'marshmallow.fields.Raw'>, <class 'int'>: <class 'marshmallow.fields.Integer'>,
<class 'uuid.UUID'>: <class 'marshmallow.fields.UUID'>, <class 'datetime.time'>:
<class 'marshmallow.fields.Time'>, <class 'datetime.date'>: <class
'marshmallow.fields.Date'>, <class 'datetime.timedelta'>: <class
'marshmallow.fields.TimeDelta'>, <class 'decimal.Decimal'>: <class
'marshmallow.fields.Decimal'>}
```

property `dict_class`: `type`

`dump(obj: Any, *, many: bool | None = None)`

Serialize an object to native Python data types according to this *Schema's* fields.

Parameters

- `obj` – The object to serialize.
- `many` – Whether to serialize *obj* as a collection. If *None*, the value for *self.many* is used.

Returns

Serialized data

New in version 1.0.0.

Changed in version 3.0.0b7: This method returns the serialized data rather than a `(data, errors)` tuple. A `ValidationError` is raised if *obj* is invalid.

Changed in version 3.0.0rc9: Validation no longer occurs upon serialization.

`dump_fields`: `Dict[str, ma_fields.Field]`

dump*s*(*obj*: Any, **args*, *many*: bool | None = None, ***kwargs*)

Same as *dump*(*C*), except return a JSON-encoded string.

Parameters

- **obj** – The object to serialize.
- **many** – Whether to serialize *obj* as a collection. If *None*, the value for *self.many* is used.

Returns

A json string

New in version 1.0.0.

Changed in version 3.0.0b7: This method returns the serialized data rather than a (data, errors) tuple. A `ValidationError` is raised if *obj* is invalid.

error_messages: Dict[str, str] = {'unknown': 'Unknown file. See https://nitpick.rtf.d.io/en/latest/nitpick_section.html#nitpick-files.'}
Overrides for default schema-level error messages

exclude: set[Any] | MutableSet[Any]

fields: Dict[str, ma_fields.Field]

Dictionary mapping field_names -> Field objects

classmethod from_dict(*fields*: dict[str, marshmallow.fields.Field | type], *, *name*: str = 'GeneratedSchema') → type

Generate a *Schema* class given a dictionary of fields.

```
from marshmallow import Schema, fields

PersonSchema = Schema.from_dict({"name": fields.Str()})
print(PersonSchema().load({"name": "David"})) # => {'name': 'David'}
```

Generated schemas are not added to the class registry and therefore cannot be referred to by name in *Nested* fields.

Parameters

- **fields** (*dict*) – Dictionary mapping field names to field instances.
- **name** (*str*) – Optional name for the class, which will appear in the repr for the class.

New in version 3.0.0.

get_attribute(*obj*: Any, *attr*: str, *default*: Any)

Defines how to pull values from an object to serialize.

New in version 2.0.0.

Changed in version 3.0.0a1: Changed position of *obj* and *attr*.

handle_error(*error*: ValidationError, *data*: Any, *, *many*: bool, ***kwargs*)

Custom error handler function for the schema.

Parameters

- **error** – The *ValidationError* raised during (de)serialization.
- **data** – The original input data.
- **many** – Value of *many* on dump or load.

- **partial** – Value of `partial` on load.

New in version 2.0.0.

Changed in version 3.0.0rc9: Receives *many* and *partial* (on deserialization) as keyword arguments.

load(*data*: *Mapping*[*str*, *Any*] | *Iterable*[*Mapping*[*str*, *Any*]], *, *many*: *bool* | *None* = *None*, *partial*: *bool* | *Sequence*[*str*] | *AbstractSet*[*str*] | *None* = *None*, *unknown*: *str* | *None* = *None*)

Deserialize a data structure to an object defined by this Schema's fields.

Parameters

- **data** – The data to deserialize.
- **many** – Whether to deserialize *data* as a collection. If *None*, the value for *self.many* is used.
- **partial** – Whether to ignore missing fields and not require any fields declared. Propagates down to *Nested* fields as well. If its value is an iterable, only missing fields listed in that iterable will be ignored. Use dot delimiters to specify nested fields.
- **unknown** – Whether to exclude, include, or raise an error for unknown fields in the data. Use *EXCLUDE*, *INCLUDE* or *RAISE*. If *None*, the value for *self.unknown* is used.

Returns

Serialized data

New in version 1.0.0.

Changed in version 3.0.0b7: This method returns the deserialized data rather than a (data, errors) tuple. A *ValidationError* is raised if invalid data are passed.

load_fields: *Dict*[*str*, *ma_fields.Field*]

loads(*json_data*: *str*, *, *many*: *bool* | *None* = *None*, *partial*: *bool* | *Sequence*[*str*] | *AbstractSet*[*str*] | *None* = *None*, *unknown*: *str* | *None* = *None*, ***kwargs*)

Same as *load()*, except it takes a JSON string as input.

Parameters

- **json_data** – A JSON string of the data to deserialize.
- **many** – Whether to deserialize *obj* as a collection. If *None*, the value for *self.many* is used.
- **partial** – Whether to ignore missing fields and not require any fields declared. Propagates down to *Nested* fields as well. If its value is an iterable, only missing fields listed in that iterable will be ignored. Use dot delimiters to specify nested fields.
- **unknown** – Whether to exclude, include, or raise an error for unknown fields in the data. Use *EXCLUDE*, *INCLUDE* or *RAISE*. If *None*, the value for *self.unknown* is used.

Returns

Serialized data

New in version 1.0.0.

Changed in version 3.0.0b7: This method returns the deserialized data rather than a (data, errors) tuple. A *ValidationError* is raised if invalid data are passed.

on_bind_field(*field_name*: *str*, *field_obj*: *Field*) → *None*

Hook to modify a field when it is bound to the *Schema*.

No-op by default.

opts: *SchemaOpts* = <marshmallow.schema.SchemaOpts object>

set_class

alias of `OrderedSet`

validate(*data*: *Mapping*[*str*, *Any*] | *Iterable*[*Mapping*[*str*, *Any*]], *, *many*: *bool* | *None* = *None*, *partial*: *bool* | *Sequence*[*str*] | *AbstractSet*[*str*] | *None* = *None*) → *dict*[*str*, *list*[*str*]]

Validate *data* against the schema, returning a dictionary of validation errors.

Parameters

- **data** – The data to validate.
- **many** – Whether to validate *data* as a collection. If *None*, the value for *self.many* is used.
- **partial** – Whether to ignore missing fields and not require any fields declared. Propagates down to *Nested* fields as well. If its value is an iterable, only missing fields listed in that iterable will be ignored. Use dot delimiters to specify nested fields.

Returns

A dictionary of validation errors.

New in version 1.1.0.

```
class nitpick.schemas.NitpickMetaSchema(*, only: Sequence[str] | AbstractSet[str] | None = None,
exclude: Sequence[str] | AbstractSet[str] = (), many: bool =
False, context: dict | None = None, load_only: Sequence[str] |
AbstractSet[str] = (), dump_only: Sequence[str] |
AbstractSet[str] = (), partial: bool | Sequence[str] |
AbstractSet[str] | None = None, unknown: str | None = None)
```

Bases: [BaseNitpickSchema](#)

Meta info about a specific TOML style file.

class Meta

Bases: `object`

Options object for a Schema.

Example usage:

```
class Meta:
    fields = ("id", "email", "date_created")
    exclude = ("password", "secret_attribute")
```

Available options:

- **fields**: Tuple or list of fields to include in the serialized result.
- **additional**: **Tuple or list of fields to include in addition to the** explicitly declared fields. **additional** and **fields** are mutually-exclusive options.
- **include**: **Dictionary of additional fields to include in the schema. It is** usually better to define fields as class variables, but you may need to use this option, e.g., if your fields are Python keywords. May be an *OrderedDict*.
- **exclude**: **Tuple or list of fields to exclude in the serialized result.** Nested fields can be represented with dot delimiters.
- **dateformat**: Default format for *Date* <*fields.Date*> fields.
- **datetimeformat**: Default format for *DateTime* <*fields.DateTime*> fields.
- **timeformat**: Default format for *Time* <*fields.Time*> fields.

- **render_module:** Module to use for *loads* <*Schema.loads*> and *dumps* <*Schema.dumps*>. Defaults to *json* from the standard library.
- **ordered:** If *True*, output of *Schema.dump* will be a *collections.OrderedDict*.
- **index_errors:** If *True*, errors dictionaries will include the index of invalid items in a collection.
- **load_only:** Tuple or list of fields to exclude from serialized results.
- **dump_only:** Tuple or list of fields to exclude from deserialization
- **unknown:** Whether to exclude, include, or raise an error for unknown fields in the data. Use *EXCLUDE*, *INCLUDE* or *RAISE*.
- **register:** Whether to register the *Schema* with marshmallow's internal class registry. Must be *True* if you intend to refer to this *Schema* by class name in *Nested* fields. Only set this to *False* when memory usage is critical. Defaults to *True*.

OPTIONS_CLASS

alias of *SchemaOpts*

```
TYPE_MAPPING: Dict[type, Type[ma_fields.Field]] = {<class 'str'>: <class
'marshmallow.fields.String'>, <class 'bytes'>: <class 'marshmallow.fields.String'>,
<class 'datetime.datetime'>: <class 'marshmallow.fields.DateTime'>, <class
'float'>: <class 'marshmallow.fields.Float'>, <class 'bool'>: <class
'marshmallow.fields.Boolean'>, <class 'tuple'>: <class 'marshmallow.fields.Raw'>,
<class 'list'>: <class 'marshmallow.fields.Raw'>, <class 'set'>: <class
'marshmallow.fields.Raw'>, <class 'int'>: <class 'marshmallow.fields.Integer'>,
<class 'uuid.UUID'>: <class 'marshmallow.fields.UUID'>, <class 'datetime.time'>:
<class 'marshmallow.fields.Time'>, <class 'datetime.date'>: <class
'marshmallow.fields.Date'>, <class 'datetime.timedelta'>: <class
'marshmallow.fields.TimeDelta'>, <class 'decimal.Decimal'>: <class
'marshmallow.fields.Decimal'>}
```

property dict_class: *type*

dump(*obj*: Any, *, *many*: bool | None = None)

Serialize an object to native Python data types according to this Schema's fields.

Parameters

- **obj** – The object to serialize.
- **many** – Whether to serialize *obj* as a collection. If *None*, the value for *self.many* is used.

Returns

Serialized data

New in version 1.0.0.

Changed in version 3.0.0b7: This method returns the serialized data rather than a (data, errors) tuple. A *ValidationError* is raised if *obj* is invalid.

Changed in version 3.0.0rc9: Validation no longer occurs upon serialization.

dump_fields: Dict[str, ma_fields.Field]

dumps(*obj*: Any, *args, *many*: bool | None = None, **kwargs)

Same as *dump()*, except return a JSON-encoded string.

Parameters

- **obj** – The object to serialize.
- **many** – Whether to serialize *obj* as a collection. If *None*, the value for *self.many* is used.

Returns

A json string

New in version 1.0.0.

Changed in version 3.0.0b7: This method returns the serialized data rather than a (data, errors) tuple. A `ValidationError` is raised if *obj* is invalid.

error_messages: `Dict[str, str] = {'unknown': 'Unknown configuration. See https://nitpick.rtfid.io/en/latest/nitpick_section.html.'}`

Overrides for default schema-level error messages

exclude: `set[Any] | MutableSet[Any]`

fields: `Dict[str, ma_fields.Field]`

Dictionary mapping field_names -> Field objects

classmethod from_dict(*fields: dict[str, marshmallow.fields.Field | type], *, name: str = 'GeneratedSchema') → type*

Generate a *Schema* class given a dictionary of fields.

```
from marshmallow import Schema, fields

PersonSchema = Schema.from_dict({"name": fields.Str()})
print(PersonSchema().load({"name": "David"})) # => {'name': 'David'}
```

Generated schemas are not added to the class registry and therefore cannot be referred to by name in *Nested* fields.

Parameters

- **fields** (*dict*) – Dictionary mapping field names to field instances.
- **name** (*str*) – Optional name for the class, which will appear in the repr for the class.

New in version 3.0.0.

get_attribute(*obj: Any, attr: str, default: Any*)

Defines how to pull values from an object to serialize.

New in version 2.0.0.

Changed in version 3.0.0a1: Changed position of *obj* and *attr*.

handle_error(*error: ValidationError, data: Any, *, many: bool, **kwargs*)

Custom error handler function for the schema.

Parameters

- **error** – The *ValidationError* raised during (de)serialization.
- **data** – The original input data.
- **many** – Value of *many* on dump or load.
- **partial** – Value of *partial* on load.

New in version 2.0.0.

Changed in version 3.0.0rc9: Receives *many* and *partial* (on deserialization) as keyword arguments.

load(*data*: *Mapping*[*str*, *Any*] | *Iterable*[*Mapping*[*str*, *Any*]], *, *many*: *bool* | *None* = *None*, *partial*: *bool* | *Sequence*[*str*] | *AbstractSet*[*str*] | *None* = *None*, *unknown*: *str* | *None* = *None*)

Deserialize a data structure to an object defined by this Schema's fields.

Parameters

- **data** – The data to deserialize.
- **many** – Whether to deserialize *data* as a collection. If *None*, the value for *self.many* is used.
- **partial** – Whether to ignore missing fields and not require any fields declared. Propagates down to *Nested* fields as well. If its value is an iterable, only missing fields listed in that iterable will be ignored. Use dot delimiters to specify nested fields.
- **unknown** – Whether to exclude, include, or raise an error for unknown fields in the data. Use *EXCLUDE*, *INCLUDE* or *RAISE*. If *None*, the value for *self.unknown* is used.

Returns

Deserialized data

New in version 1.0.0.

Changed in version 3.0.0b7: This method returns the deserialized data rather than a (data, errors) tuple. A `ValidationError` is raised if invalid data are passed.

load_fields: `Dict[str, ma_fields.Field]`

loads(*json_data*: *str*, *, *many*: *bool* | *None* = *None*, *partial*: *bool* | *Sequence*[*str*] | *AbstractSet*[*str*] | *None* = *None*, *unknown*: *str* | *None* = *None*, ***kwargs*)

Same as `load()`, except it takes a JSON string as input.

Parameters

- **json_data** – A JSON string of the data to deserialize.
- **many** – Whether to deserialize *obj* as a collection. If *None*, the value for *self.many* is used.
- **partial** – Whether to ignore missing fields and not require any fields declared. Propagates down to *Nested* fields as well. If its value is an iterable, only missing fields listed in that iterable will be ignored. Use dot delimiters to specify nested fields.
- **unknown** – Whether to exclude, include, or raise an error for unknown fields in the data. Use *EXCLUDE*, *INCLUDE* or *RAISE*. If *None*, the value for *self.unknown* is used.

Returns

Deserialized data

New in version 1.0.0.

Changed in version 3.0.0b7: This method returns the deserialized data rather than a (data, errors) tuple. A `ValidationError` is raised if invalid data are passed.

on_bind_field(*field_name*: *str*, *field_obj*: `Field`) → *None*

Hook to modify a field when it is bound to the *Schema*.

No-op by default.

opts: `SchemaOpts` = `<marshmallow.schema.SchemaOpts object>`

set_class

alias of `OrderedSet`

```
validate(data: Mapping[str, Any] | Iterable[Mapping[str, Any]], *, many: bool | None = None, partial: bool | Sequence[str] | AbstractSet[str] | None = None) → dict[str, list[str]]
```

Validate *data* against the schema, returning a dictionary of validation errors.

Parameters

- **data** – The data to validate.
- **many** – Whether to validate *data* as a collection. If *None*, the value for *self.many* is used.
- **partial** – Whether to ignore missing fields and not require any fields declared. Propagates down to *Nested* fields as well. If its value is an iterable, only missing fields listed in that iterable will be ignored. Use dot delimiters to specify nested fields.

Returns

A dictionary of validation errors.

New in version 1.1.0.

```
class nitpick.schemas.NitpickSectionSchema(*, only: Sequence[str] | AbstractSet[str] | None = None,
exclude: Sequence[str] | AbstractSet[str] = (), many: bool =
False, context: dict | None = None, load_only: Sequence[str] |
AbstractSet[str] = (), dump_only: Sequence[str] |
AbstractSet[str] = (), partial: bool | Sequence[str] |
AbstractSet[str] | None = None, unknown: str | None =
None)
```

Bases: [BaseNitpickSchema](#)

Validation schema for the [nitpick] section on the style file.

class Meta

Bases: `object`

Options object for a Schema.

Example usage:

```
class Meta:
    fields = ("id", "email", "date_created")
    exclude = ("password", "secret_attribute")
```

Available options:

- **fields**: Tuple or list of fields to include in the serialized result.
- **additional**: Tuple or list of fields to include in addition to the explicitly declared fields. **additional** and **fields** are mutually-exclusive options.
- **include**: Dictionary of additional fields to include in the schema. It is usually better to define fields as class variables, but you may need to use this option, e.g., if your fields are Python keywords. May be an *OrderedDict*.
- **exclude**: Tuple or list of fields to exclude in the serialized result. Nested fields can be represented with dot delimiters.
- **dateformat**: Default format for *Date* <fields.Date> fields.
- **datetimeformat**: Default format for *DateTime* <fields.DateTime> fields.
- **timeformat**: Default format for *Time* <fields.Time> fields.
- **render_module**: Module to use for loads <Schema.loads> and dumps <Schema.dumps>. Defaults to *json* from the standard library.

- **ordered**: If *True*, output of *Schema.dump* will be a *collections.OrderedDict*.
- **index_errors**: If *True*, errors dictionaries will include the index of invalid items in a collection.
- **load_only**: Tuple or list of fields to exclude from serialized results.
- **dump_only**: Tuple or list of fields to exclude from deserialization
- **unknown**: Whether to exclude, include, or raise an error for unknown fields in the data. Use *EXCLUDE*, *INCLUDE* or *RAISE*.
- **register**: Whether to register the *Schema* with marshmallow's internal class registry. Must be *True* if you intend to refer to this *Schema* by class name in *Nested* fields. Only set this to *False* when memory usage is critical. Defaults to *True*.

OPTIONS_CLASS

alias of [SchemaOpts](#)

```
TYPE_MAPPING: Dict[type, Type[ma_fields.Field]] = {<class 'str'>: <class
'marshmallow.fields.String'>, <class 'bytes'>: <class 'marshmallow.fields.String'>,
<class 'datetime.datetime'>: <class 'marshmallow.fields.Date'>, <class
'float'>: <class 'marshmallow.fields.Float'>, <class 'bool'>: <class
'marshmallow.fields.Boolean'>, <class 'tuple'>: <class 'marshmallow.fields.Raw'>,
<class 'list'>: <class 'marshmallow.fields.Raw'>, <class 'set'>: <class
'marshmallow.fields.Raw'>, <class 'int'>: <class 'marshmallow.fields.Integer'>,
<class 'uuid.UUID'>: <class 'marshmallow.fields.UUID'>, <class 'datetime.time'>:
<class 'marshmallow.fields.Time'>, <class 'datetime.date'>: <class
'marshmallow.fields.Date'>, <class 'datetime.timedelta'>: <class
'marshmallow.fields.TimeDelta'>, <class 'decimal.Decimal'>: <class
'marshmallow.fields.Decimal'>}
```

property dict_class: [type](#)

dump(obj: Any, *, many: bool | None = None)

Serialize an object to native Python data types according to this Schema's fields.

Parameters

- **obj** – The object to serialize.
- **many** – Whether to serialize *obj* as a collection. If *None*, the value for *self.many* is used.

Returns

Serialized data

New in version 1.0.0.

Changed in version 3.0.0b7: This method returns the serialized data rather than a (data, errors) tuple. A [ValidationError](#) is raised if *obj* is invalid.

Changed in version 3.0.0rc9: Validation no longer occurs upon serialization.

dump_fields: [Dict\[str, ma_fields.Field\]](#)

dumps(obj: Any, *args, many: bool | None = None, **kwargs)

Same as [dump\(\)](#), except return a JSON-encoded string.

Parameters

- **obj** – The object to serialize.
- **many** – Whether to serialize *obj* as a collection. If *None*, the value for *self.many* is used.

Returns

A json string

New in version 1.0.0.

Changed in version 3.0.0b7: This method returns the serialized data rather than a (data, errors) tuple. A `ValidationError` is raised if obj is invalid.

error_messages: `Dict[str, str] = {'unknown': 'Unknown configuration. See https://nitpick.rtf.d.io/en/latest/nitpick_section.html.'}`

Overrides for default schema-level error messages

exclude: `set[Any] | MutableSet[Any]`

fields: `Dict[str, ma_fields.Field]`

Dictionary mapping field_names -> Field objects

classmethod from_dict(fields: dict[str, marshmallow.fields.Field | type], *, name: str = 'GeneratedSchema') -> type

Generate a *Schema* class given a dictionary of fields.

```
from marshmallow import Schema, fields

PersonSchema = Schema.from_dict({"name": fields.Str()})
print(PersonSchema().load({"name": "David"})) # => {'name': 'David'}
```

Generated schemas are not added to the class registry and therefore cannot be referred to by name in *Nested* fields.

Parameters

- **fields** (*dict*) – Dictionary mapping field names to field instances.
- **name** (*str*) – Optional name for the class, which will appear in the repr for the class.

New in version 3.0.0.

get_attribute(obj: Any, attr: str, default: Any)

Defines how to pull values from an object to serialize.

New in version 2.0.0.

Changed in version 3.0.0a1: Changed position of obj and attr.

handle_error(error: ValidationError, data: Any, *, many: bool, **kwargs)

Custom error handler function for the schema.

Parameters

- **error** – The *ValidationError* raised during (de)serialization.
- **data** – The original input data.
- **many** – Value of many on dump or load.
- **partial** – Value of partial on load.

New in version 2.0.0.

Changed in version 3.0.0rc9: Receives *many* and *partial* (on deserialization) as keyword arguments.

load(*data*: *Mapping*[*str*, *Any*] | *Iterable*[*Mapping*[*str*, *Any*]], *, *many*: *bool* | *None* = *None*, *partial*: *bool* | *Sequence*[*str*] | *AbstractSet*[*str*] | *None* = *None*, *unknown*: *str* | *None* = *None*)

Deserialize a data structure to an object defined by this Schema's fields.

Parameters

- **data** – The data to deserialize.
- **many** – Whether to deserialize *data* as a collection. If *None*, the value for *self.many* is used.
- **partial** – Whether to ignore missing fields and not require any fields declared. Propagates down to *Nested* fields as well. If its value is an iterable, only missing fields listed in that iterable will be ignored. Use dot delimiters to specify nested fields.
- **unknown** – Whether to exclude, include, or raise an error for unknown fields in the data. Use *EXCLUDE*, *INCLUDE* or *RAISE*. If *None*, the value for *self.unknown* is used.

Returns

Deserialized data

New in version 1.0.0.

Changed in version 3.0.0b7: This method returns the deserialized data rather than a (data, errors) tuple. A `ValidationError` is raised if invalid data are passed.

load_fields: `Dict[str, ma_fields.Field]`

loads(*json_data*: *str*, *, *many*: *bool* | *None* = *None*, *partial*: *bool* | *Sequence*[*str*] | *AbstractSet*[*str*] | *None* = *None*, *unknown*: *str* | *None* = *None*, ***kwargs*)

Same as `load()`, except it takes a JSON string as input.

Parameters

- **json_data** – A JSON string of the data to deserialize.
- **many** – Whether to deserialize *obj* as a collection. If *None*, the value for *self.many* is used.
- **partial** – Whether to ignore missing fields and not require any fields declared. Propagates down to *Nested* fields as well. If its value is an iterable, only missing fields listed in that iterable will be ignored. Use dot delimiters to specify nested fields.
- **unknown** – Whether to exclude, include, or raise an error for unknown fields in the data. Use *EXCLUDE*, *INCLUDE* or *RAISE*. If *None*, the value for *self.unknown* is used.

Returns

Deserialized data

New in version 1.0.0.

Changed in version 3.0.0b7: This method returns the deserialized data rather than a (data, errors) tuple. A `ValidationError` is raised if invalid data are passed.

on_bind_field(*field_name*: *str*, *field_obj*: `Field`) → *None*

Hook to modify a field when it is bound to the *Schema*.

No-op by default.

opts: `SchemaOpts` = `<marshmallow.schema.SchemaOpts object>`

set_class

alias of `OrderedSet`

```
validate(data: Mapping[str, Any] | Iterable[Mapping[str, Any]], *, many: bool | None = None, partial: bool | Sequence[str] | AbstractSet[str] | None = None) → dict[str, list[str]]
```

Validate *data* against the schema, returning a dictionary of validation errors.

Parameters

- **data** – The data to validate.
- **many** – Whether to validate *data* as a collection. If *None*, the value for *self.many* is used.
- **partial** – Whether to ignore missing fields and not require any fields declared. Propagates down to *Nested* fields as well. If its value is an iterable, only missing fields listed in that iterable will be ignored. Use dot delimiters to specify nested fields.

Returns

A dictionary of validation errors.

New in version 1.1.0.

```
class nitpick.schemas.NitpickStylesSectionSchema(*, only: Sequence[str] | AbstractSet[str] | None = None, exclude: Sequence[str] | AbstractSet[str] = (), many: bool = False, context: dict | None = None, load_only: Sequence[str] | AbstractSet[str] = (), dump_only: Sequence[str] | AbstractSet[str] = (), partial: bool | Sequence[str] | AbstractSet[str] | None = None, unknown: str | None = None)
```

Bases: [BaseNitpickSchema](#)

Validation schema for the [nitpick.styles] section on the style file.

class Meta

Bases: `object`

Options object for a Schema.

Example usage:

```
class Meta:
    fields = ("id", "email", "date_created")
    exclude = ("password", "secret_attribute")
```

Available options:

- **fields**: Tuple or list of fields to include in the serialized result.
- **additional**: Tuple or list of fields to include in addition to the explicitly declared fields. **additional** and **fields** are mutually-exclusive options.
- **include**: Dictionary of additional fields to include in the schema. It is usually better to define fields as class variables, but you may need to use this option, e.g., if your fields are Python keywords. May be an *OrderedDict*.
- **exclude**: Tuple or list of fields to exclude in the serialized result. Nested fields can be represented with dot delimiters.
- **dateformat**: Default format for *Date* <fields.Date> fields.
- **datetimeformat**: Default format for *DateTime* <fields.DateTime> fields.
- **timeformat**: Default format for *Time* <fields.Time> fields.
- **render_module**: Module to use for loads <Schema.loads> and dumps <Schema.dumps>. Defaults to *json* from the standard library.

- **ordered**: If *True*, output of *Schema.dump* will be a *collections.OrderedDict*.
- **index_errors**: If *True*, errors dictionaries will include the index of invalid items in a collection.
- **load_only**: Tuple or list of fields to exclude from serialized results.
- **dump_only**: Tuple or list of fields to exclude from deserialization
- **unknown**: Whether to exclude, include, or raise an error for unknown fields in the data. Use *EXCLUDE*, *INCLUDE* or *RAISE*.
- **register**: Whether to register the *Schema* with marshmallow's internal class registry. Must be *True* if you intend to refer to this *Schema* by class name in *Nested* fields. Only set this to *False* when memory usage is critical. Defaults to *True*.

OPTIONS_CLASS

alias of [SchemaOpts](#)

```
TYPE_MAPPING: Dict[type, Type[ma_fields.Field]] = {<class 'str'>: <class
'marshmallow.fields.String'>, <class 'bytes'>: <class 'marshmallow.fields.String'>,
<class 'datetime.datetime'>: <class 'marshmallow.fields.Date'>, <class
'float'>: <class 'marshmallow.fields.Float'>, <class 'bool'>: <class
'marshmallow.fields.Boolean'>, <class 'tuple'>: <class 'marshmallow.fields.Raw'>,
<class 'list'>: <class 'marshmallow.fields.Raw'>, <class 'set'>: <class
'marshmallow.fields.Raw'>, <class 'int'>: <class 'marshmallow.fields.Integer'>,
<class 'uuid.UUID'>: <class 'marshmallow.fields.UUID'>, <class 'datetime.time'>:
<class 'marshmallow.fields.Time'>, <class 'datetime.date'>: <class
'marshmallow.fields.Date'>, <class 'datetime.timedelta'>: <class
'marshmallow.fields.TimeDelta'>, <class 'decimal.Decimal'>: <class
'marshmallow.fields.Decimal'>}
```

property dict_class: [type](#)

dump(obj: Any, *, many: bool | None = None)

Serialize an object to native Python data types according to this Schema's fields.

Parameters

- **obj** – The object to serialize.
- **many** – Whether to serialize *obj* as a collection. If *None*, the value for *self.many* is used.

Returns

Serialized data

New in version 1.0.0.

Changed in version 3.0.0b7: This method returns the serialized data rather than a (data, errors) tuple. A [ValidationError](#) is raised if *obj* is invalid.

Changed in version 3.0.0rc9: Validation no longer occurs upon serialization.

dump_fields: [Dict\[str, ma_fields.Field\]](#)

dumps(obj: Any, *args, many: bool | None = None, **kwargs)

Same as [dump\(\)](#), except return a JSON-encoded string.

Parameters

- **obj** – The object to serialize.
- **many** – Whether to serialize *obj* as a collection. If *None*, the value for *self.many* is used.

Returns

A json string

New in version 1.0.0.

Changed in version 3.0.0b7: This method returns the serialized data rather than a (data, errors) tuple. A `ValidationError` is raised if `obj` is invalid.

error_messages: `Dict[str, str] = {'unknown': 'Unknown configuration. See https://nitpick.rtf.d.io/en/latest/nitpick_section.html#nitpick-styles.'}`

Overrides for default schema-level error messages

exclude: `set[Any] | MutableSet[Any]`

fields: `Dict[str, ma_fields.Field]`

Dictionary mapping field_names -> Field objects

classmethod from_dict(fields: dict[str, marshmallow.fields.Field | type], *, name: str = 'GeneratedSchema') -> type

Generate a *Schema* class given a dictionary of fields.

```
from marshmallow import Schema, fields

PersonSchema = Schema.from_dict({"name": fields.Str()})
print(PersonSchema().load({"name": "David"})) # => {'name': 'David'}
```

Generated schemas are not added to the class registry and therefore cannot be referred to by name in *Nested* fields.

Parameters

- **fields** (*dict*) – Dictionary mapping field names to field instances.
- **name** (*str*) – Optional name for the class, which will appear in the repr for the class.

New in version 3.0.0.

get_attribute(obj: Any, attr: str, default: Any)

Defines how to pull values from an object to serialize.

New in version 2.0.0.

Changed in version 3.0.0a1: Changed position of `obj` and `attr`.

handle_error(error: ValidationError, data: Any, *, many: bool, **kwargs)

Custom error handler function for the schema.

Parameters

- **error** – The *ValidationError* raised during (de)serialization.
- **data** – The original input data.
- **many** – Value of `many` on dump or load.
- **partial** – Value of `partial` on load.

New in version 2.0.0.

Changed in version 3.0.0rc9: Receives *many* and *partial* (on deserialization) as keyword arguments.

load(*data*: *Mapping*[*str*, *Any*] | *Iterable*[*Mapping*[*str*, *Any*]], *, *many*: *bool* | *None* = *None*, *partial*: *bool* | *Sequence*[*str*] | *AbstractSet*[*str*] | *None* = *None*, *unknown*: *str* | *None* = *None*)

Deserialize a data structure to an object defined by this Schema's fields.

Parameters

- **data** – The data to deserialize.
- **many** – Whether to deserialize *data* as a collection. If *None*, the value for *self.many* is used.
- **partial** – Whether to ignore missing fields and not require any fields declared. Propagates down to *Nested* fields as well. If its value is an iterable, only missing fields listed in that iterable will be ignored. Use dot delimiters to specify nested fields.
- **unknown** – Whether to exclude, include, or raise an error for unknown fields in the data. Use *EXCLUDE*, *INCLUDE* or *RAISE*. If *None*, the value for *self.unknown* is used.

Returns

Deserialized data

New in version 1.0.0.

Changed in version 3.0.0b7: This method returns the deserialized data rather than a (data, errors) tuple. A `ValidationError` is raised if invalid data are passed.

load_fields: `Dict[str, ma_fields.Field]`

loads(*json_data*: *str*, *, *many*: *bool* | *None* = *None*, *partial*: *bool* | *Sequence*[*str*] | *AbstractSet*[*str*] | *None* = *None*, *unknown*: *str* | *None* = *None*, ***kwargs*)

Same as `load()`, except it takes a JSON string as input.

Parameters

- **json_data** – A JSON string of the data to deserialize.
- **many** – Whether to deserialize *obj* as a collection. If *None*, the value for *self.many* is used.
- **partial** – Whether to ignore missing fields and not require any fields declared. Propagates down to *Nested* fields as well. If its value is an iterable, only missing fields listed in that iterable will be ignored. Use dot delimiters to specify nested fields.
- **unknown** – Whether to exclude, include, or raise an error for unknown fields in the data. Use *EXCLUDE*, *INCLUDE* or *RAISE*. If *None*, the value for *self.unknown* is used.

Returns

Deserialized data

New in version 1.0.0.

Changed in version 3.0.0b7: This method returns the deserialized data rather than a (data, errors) tuple. A `ValidationError` is raised if invalid data are passed.

on_bind_field(*field_name*: *str*, *field_obj*: `Field`) → *None*

Hook to modify a field when it is bound to the *Schema*.

No-op by default.

opts: `SchemaOpts` = `<marshmallow.schema.SchemaOpts object>`

set_class

alias of `OrderedSet`

validate(*data*: *Mapping*[*str*, *Any*] | *Iterable*[*Mapping*[*str*, *Any*]], *, *many*: *bool* | *None* = *None*, *partial*: *bool* | *Sequence*[*str*] | *AbstractSet*[*str*] | *None* = *None*) → *dict*[*str*, *list*[*str*]]

Validate *data* against the schema, returning a dictionary of validation errors.

Parameters

- **data** – The data to validate.
- **many** – Whether to validate *data* as a collection. If *None*, the value for *self.many* is used.
- **partial** – Whether to ignore missing fields and not require any fields declared. Propagates down to *Nested* fields as well. If its value is an iterable, only missing fields listed in that iterable will be ignored. Use dot delimiters to specify nested fields.

Returns

A dictionary of validation errors.

New in version 1.1.0.

`nitpick.schemas.flatten_marshmallow_errors(errors: dict) → str`

Flatten Marshmallow errors to a string.

`nitpick.schemas.help_message(sentence: str, help_page: str) → str`

Show help with the documentation URL on validation errors.

nitpick.style module

Style parsing and merging.

class `nitpick.style.BuiltinStyle`(*, *formatted*: *str*, *path_from_resources_root*: *str*)

Bases: `object`

A built-in style file in TOML format.

Method generated by attrs for class `BuiltinStyle`.

files: `list[str]`

formatted: `str`

classmethod `from_path`(*resource_path*: *Path*, *library_dir*: *Path* | *None* = *None*) → *BuiltinStyle*

Create a style from its path.

identify_tag: `str`

name: `str`

path_from_resources_root: `str`

url: `str`

class `nitpick.style.ConfigValidator`(*project*: *Project*)

Bases: `object`

Validate a nitpick configuration.

project: *Project*

validate(*config_dict*: *dict*) → *tuple*[*dict*, *dict*]

Validate an already parsed toml file.

```
class nitpick.style.FileFetcher(session: ~requests_cache.session.CachedSession | None = None, protocols:
    tuple[str, ...] = (<Scheme.FILE: 'file'>, ), domains: tuple[str, ...] = ())
```

Bases: *StyleFetcher*

Fetch a style from a local file.

domains: tuple[str, ...] = ()

fetch(*url*: furl) → str

Fetch a style from a local file.

normalize(*url*: furl) → furl

Normalize a URL.

Produces a canonical URL, meant to be used to uniquely identify a style resource.

- The base name has .toml appended if not already ending in that extension
- Individual fetchers can further normalize the path and scheme.

preprocess_relative_url(*url*: str) → str

Preprocess a relative URL.

Only called for urls that lack a scheme (at the very least), being resolved against a base URL that matches this specific fetcher.

Relative paths are file paths; any ~ home reference is expanded at this point.

protocols: tuple[str, ...] = (<Scheme.FILE: 'file'>,))

requires_connection: ClassVar[bool] = False

session: CachedSession | None = None

```
class nitpick.style.GitHubFetcher(session: ~requests_cache.session.CachedSession | None = None,
    protocols: tuple[str, ...] = (<Scheme.GH: 'gh'>, <Scheme.GITHUB:
    'github'>), domains: tuple[str, ...] = ('github.com', ))
```

Bases: *HttpFetcher*

Fetch styles from GitHub repositories.

domains: tuple[str, ...] = ('github.com',)

fetch(*url*: furl) → str

Fetch the style from a web location.

normalize(*url*: furl) → furl

Normalize a URL.

Produces a canonical URL, meant to be used to uniquely identify a style resource.

- The base name has .toml appended if not already ending in that extension
- Individual fetchers can further normalize the path and scheme.

preprocess_relative_url(*url*: str) → str

Preprocess a relative URL.

Only called for urls that lack a scheme (at the very least), being resolved against a base URL that matches this specific fetcher.

protocols: tuple[str, ...] = (<Scheme.GH: 'gh'>, <Scheme.GITHUB: 'github'>)

requires_connection: `ClassVar[bool] = True`

session: `CachedSession | None = None`

```
class nitpick.style.GitHubURL(owner: str, repository: str, git_reference: str, path: tuple[str, ...] = (),
                             auth_token: str | None = None, query_params: tuple[tuple[str, str], ...] |
                             None = None)
```

Bases: `object`

Represent a GitHub URL, created from a URL or from its parts.

property api_url: `furl`

API URL for this repo.

auth_token: `str | None = None`

property authorization_header: `dict[str, str] | None`

Authorization header encoded in this URL.

property default_branch: `str`

Default GitHub branch.

classmethod from_furl(url: furl) → *GitHubURL*

Create an instance from a parsed URL in any accepted format.

See the code for `test_parsing_github_urls()` for more examples.

git_reference: `str`

property git_reference_or_default: `str`

Return the Git reference if informed, or return the default branch.

property long_protocol_url: `furl`

Long protocol URL (github).

owner: `str`

path: `tuple[str, ...] = ()`

query_params: `tuple[tuple[str, str], ...] | None = None`

property raw_content_url: `furl`

Raw content URL for this path.

repository: `str`

property short_protocol_url: `furl`

Short protocol URL (gh).

property token: `str | None`

Token encoded in this URL.

If present and it starts with a \$, it will be replaced with the value of the environment corresponding to the remaining part of the string.

property url: `furl`

Default URL built from attributes.


```
class nitpick.style.HttpFetcher(session: ~requests_cache.session.CachedSession | None = None,
                                protocols: tuple[str, ...] = (<Scheme.HTTP: 'http'>, <Scheme.HTTPS:
                                'https'>), domains: tuple[str, ...] = ())
```

Bases: [StyleFetcher](#)

Fetch a style from an http/https server.

domains: tuple[str, ...] = ()

fetch(*url*: furl) → str

Fetch the style from a web location.

normalize(*url*: furl) → furl

Normalize a URL.

Produces a canonical URL, meant to be used to uniquely identify a style resource.

- The base name has .toml appended if not already ending in that extension
- Individual fetchers can further normalize the path and scheme.

preprocess_relative_url(*url*: str) → str

Preprocess a relative URL.

Only called for urls that lack a scheme (at the very least), being resolved against a base URL that matches this specific fetcher.

protocols: tuple[str, ...] = (<Scheme.HTTP: 'http'>, <Scheme.HTTPS: 'https'>)

requires_connection: ClassVar[bool] = True

session: CachedSession | None = None

```
class nitpick.style.PythonPackageFetcher(session: ~requests_cache.session.CachedSession | None =
                                           None, protocols: tuple[str, ...] = (<Scheme.PY: 'py'>,
                                           <Scheme.PYPACKAGE: 'pypackage'>), domains: tuple[str, ...]
                                           = ())
```

Bases: [StyleFetcher](#)

Fetch a style from an installed Python package.

URL schemes: - py://import/path/of/style/file/<style_file_name> - pypackage://import/path/of/style/file/<style_file_name>

E.g. py://some_package/path/nitpick.toml.

domains: tuple[str, ...] = ()

fetch(*url*: furl) → str

Fetch the style from a Python package.

normalize(*url*: furl) → furl

Normalize a URL.

Produces a canonical URL, meant to be used to uniquely identify a style resource.

- The base name has .toml appended if not already ending in that extension
- Individual fetchers can further normalize the path and scheme.

preprocess_relative_url(*url: str*) → *str*

Preprocess a relative URL.

Only called for urls that lack a scheme (at the very least), being resolved against a base URL that matches this specific fetcher.

protocols: **tuple**[**str**, ...] = (<Scheme.PY: 'py'>, <Scheme.PYPACKAGE: 'pypackage'>)

requires_connection: **ClassVar**[**bool**] = **False**

session: **CachedSession** | **None** = **None**

class nitpick.style.**PythonPackageURL**(*import_path: str*, *resource_name: str*)

Bases: **object**

Represent a resource file in installed Python package.

property content_path: **Path**

Raw path of resource file.

classmethod from_furl(*url: furl*) → *PythonPackageURL*

Create an instance from a parsed URL in any accepted format.

See the code for `test_parsing_python_package_urls()` for more examples.

import_path: **str**

resource_name: **str**

class nitpick.style.**Scheme**(*value, names=None, *, module=None, qualname=None, type=None, start=1, boundary=None*)

Bases: **LowercaseStrEnum**

URL schemes.

FILE = 'file'

GH = 'gh'

GITHUB = 'github'

HTTP = 'http'

HTTPS = 'https'

PY = 'py'

PYPACKAGE = 'pypackage'

capitalize()

Return a capitalized version of the string.

More specifically, make the first character have upper case and the rest lower case.

casefold()

Return a version of the string suitable for caseless comparisons.

center(*width, fillchar=' ', /*)

Return a centered string of length width.

Padding is done using the specified fill character (default is a space).

count(*sub*[, *start*[, *end*]]) → int

Return the number of non-overlapping occurrences of substring *sub* in string *S*[*start*:*end*]. Optional arguments *start* and *end* are interpreted as in slice notation.

encode(*encoding*='utf-8', *errors*='strict')

Encode the string using the codec registered for encoding.

encoding

The encoding in which to encode the string.

errors

The error handling scheme to use for encoding errors. The default is 'strict' meaning that encoding errors raise a UnicodeEncodeError. Other possible values are 'ignore', 'replace' and 'xmlcharrefreplace' as well as any other name registered with codecs.register_error that can handle UnicodeEncodeErrors.

endswith(*suffix*[, *start*[, *end*]]) → bool

Return True if *S* ends with the specified suffix, False otherwise. With optional *start*, test *S* beginning at that position. With optional *end*, stop comparing *S* at that position. *suffix* can also be a tuple of strings to try.

expandtabs(*tabsize*=8)

Return a copy where all tab characters are expanded using spaces.

If *tabsize* is not given, a tab size of 8 characters is assumed.

find(*sub*[, *start*[, *end*]]) → int

Return the lowest index in *S* where substring *sub* is found, such that *sub* is contained within *S*[*start*:*end*]. Optional arguments *start* and *end* are interpreted as in slice notation.

Return -1 on failure.

format(**args*, ***kwargs*) → str

Return a formatted version of *S*, using substitutions from *args* and *kwargs*. The substitutions are identified by braces ('{' and '}').

format_map(*mapping*) → str

Return a formatted version of *S*, using substitutions from *mapping*. The substitutions are identified by braces ('{' and '}').

index(*sub*[, *start*[, *end*]]) → int

Return the lowest index in *S* where substring *sub* is found, such that *sub* is contained within *S*[*start*:*end*]. Optional arguments *start* and *end* are interpreted as in slice notation.

Raises ValueError when the substring is not found.

isalnum()

Return True if the string is an alpha-numeric string, False otherwise.

A string is alpha-numeric if all characters in the string are alpha-numeric and there is at least one character in the string.

isalpha()

Return True if the string is an alphabetic string, False otherwise.

A string is alphabetic if all characters in the string are alphabetic and there is at least one character in the string.

isascii()

Return True if all characters in the string are ASCII, False otherwise.

ASCII characters have code points in the range U+0000-U+007F. Empty string is ASCII too.

isdecimal()

Return True if the string is a decimal string, False otherwise.

A string is a decimal string if all characters in the string are decimal and there is at least one character in the string.

isdigit()

Return True if the string is a digit string, False otherwise.

A string is a digit string if all characters in the string are digits and there is at least one character in the string.

isidentifier()

Return True if the string is a valid Python identifier, False otherwise.

Call keyword.iskeyword(s) to test whether string s is a reserved identifier, such as “def” or “class”.

islower()

Return True if the string is a lowercase string, False otherwise.

A string is lowercase if all cased characters in the string are lowercase and there is at least one cased character in the string.

isnumeric()

Return True if the string is a numeric string, False otherwise.

A string is numeric if all characters in the string are numeric and there is at least one character in the string.

isprintable()

Return True if the string is printable, False otherwise.

A string is printable if all of its characters are considered printable in repr() or if it is empty.

isspace()

Return True if the string is a whitespace string, False otherwise.

A string is whitespace if all characters in the string are whitespace and there is at least one character in the string.

istitle()

Return True if the string is a title-cased string, False otherwise.

In a title-cased string, upper- and title-case characters may only follow uncased characters and lowercase characters only cased ones.

isupper()

Return True if the string is an uppercase string, False otherwise.

A string is uppercase if all cased characters in the string are uppercase and there is at least one cased character in the string.

join(iterable, /)

Concatenate any number of strings.

The string whose method is called is inserted in between each given string. The result is returned as a new string.

Example: `'.'.join(['ab', 'pq', 'rs']) -> 'ab.pq.rs'`

ljust(*width*, *fillchar*=' ', /)

Return a left-justified string of length *width*.

Padding is done using the specified fill character (default is a space).

lower()

Return a copy of the string converted to lowercase.

lstrip(*chars*=None, /)

Return a copy of the string with leading whitespace removed.

If *chars* is given and not None, remove characters in *chars* instead.

static maketrans()

Return a translation table usable for `str.translate()`.

If there is only one argument, it must be a dictionary mapping Unicode ordinals (integers) or characters to Unicode ordinals, strings or None. Character keys will be then converted to ordinals. If there are two arguments, they must be strings of equal length, and in the resulting dictionary, each character in *x* will be mapped to the character at the same position in *y*. If there is a third argument, it must be a string, whose characters will be mapped to None in the result.

partition(*sep*, /)

Partition the string into three parts using the given separator.

This will search for the separator in the string. If the separator is found, returns a 3-tuple containing the part before the separator, the separator itself, and the part after it.

If the separator is not found, returns a 3-tuple containing the original string and two empty strings.

removeprefix(*prefix*, /)

Return a str with the given prefix string removed if present.

If the string starts with the prefix string, return `string[len(prefix):]`. Otherwise, return a copy of the original string.

removesuffix(*suffix*, /)

Return a str with the given suffix string removed if present.

If the string ends with the suffix string and that suffix is not empty, return `string[:-len(suffix)]`. Otherwise, return a copy of the original string.

replace(*old*, *new*, *count*=-1, /)

Return a copy with all occurrences of substring *old* replaced by *new*.

count

Maximum number of occurrences to replace. -1 (the default value) means replace all occurrences.

If the optional argument *count* is given, only the first *count* occurrences are replaced.

rfind(*sub*[, *start*[, *end*]]) → int

Return the highest index in *S* where substring *sub* is found, such that *sub* is contained within *S*[*start*:*end*]. Optional arguments *start* and *end* are interpreted as in slice notation.

Return -1 on failure.

rindex(*sub*[, *start*[, *end*]]) → int

Return the highest index in *S* where substring *sub* is found, such that *sub* is contained within *S*[*start*:*end*]. Optional arguments *start* and *end* are interpreted as in slice notation.

Raises `ValueError` when the substring is not found.

rjust(*width*, *fillchar*=' ', /)

Return a right-justified string of length *width*.

Padding is done using the specified fill character (default is a space).

rpartition(*sep*, /)

Partition the string into three parts using the given separator.

This will search for the separator in the string, starting at the end. If the separator is found, returns a 3-tuple containing the part before the separator, the separator itself, and the part after it.

If the separator is not found, returns a 3-tuple containing two empty strings and the original string.

rsplit(*sep*=None, *maxsplit*=-1)

Return a list of the substrings in the string, using *sep* as the separator string.

sep

The separator used to split the string.

When set to None (the default value), will split on any whitespace character (including `n r t f` and spaces) and will discard empty strings from the result.

maxsplit

Maximum number of splits (starting from the left). -1 (the default value) means no limit.

Splitting starts at the end of the string and works to the front.

rstrip(*chars*=None, /)

Return a copy of the string with trailing whitespace removed.

If *chars* is given and not None, remove characters in *chars* instead.

split(*sep*=None, *maxsplit*=-1)

Return a list of the substrings in the string, using *sep* as the separator string.

sep

The separator used to split the string.

When set to None (the default value), will split on any whitespace character (including `n r t f` and spaces) and will discard empty strings from the result.

maxsplit

Maximum number of splits (starting from the left). -1 (the default value) means no limit.

Note, `str.split()` is mainly useful for data that has been intentionally delimited. With natural text that includes punctuation, consider using the regular expression module.

splitlines(*keepends*=False)

Return a list of the lines in the string, breaking at line boundaries.

Line breaks are not included in the resulting list unless *keepends* is given and true.

startswith(*prefix*[, *start*[, *end*]]) → bool

Return True if *S* starts with the specified prefix, False otherwise. With optional *start*, test *S* beginning at that position. With optional *end*, stop comparing *S* at that position. *prefix* can also be a tuple of strings to try.

strip(*chars*=None, /)

Return a copy of the string with leading and trailing whitespace removed.

If *chars* is given and not None, remove characters in *chars* instead.

swapcase()

Convert uppercase characters to lowercase and lowercase characters to uppercase.

title()

Return a version of the string where each word is titlecased.

More specifically, words start with uppercased characters and all remaining cased characters have lower case.

translate(*table*, /)

Replace each character in the string using the given translation table.

table

Translation table, which must be a mapping of Unicode ordinals to Unicode ordinals, strings, or None.

The table must implement lookup/indexing via `__getitem__`, for instance a dictionary or list. If this operation raises `LookupError`, the character is left untouched. Characters mapped to None are deleted.

upper()

Return a copy of the string converted to uppercase.

zfill(*width*, /)

Pad a numeric string with zeros on the left, to fill a field of the given width.

The string is never truncated.

```
class nitpick.style.StyleFetcher(session: CachedSession | None = None, protocols: tuple[str, ...] = (),
                                domains: tuple[str, ...] = ())
```

Bases: `object`

Base class of all fetchers, it encapsulates get/fetch from a specific source.

domains: `tuple[str, ...] = ()`

fetch(*url*: furl) → str

Fetch a style from a specific fetcher.

normalize(*url*: furl) → furl

Normalize a URL.

Produces a canonical URL, meant to be used to uniquely identify a style resource.

- The base name has `.toml` appended if not already ending in that extension
- Individual fetchers can further normalize the path and scheme.

preprocess_relative_url(*url*: str) → str

Preprocess a relative URL.

Only called for urls that lack a scheme (at the very least), being resolved against a base URL that matches this specific fetcher.

protocols: `tuple[str, ...] = ()`

requires_connection: `ClassVar[bool] = False`

session: `CachedSession | None = None`

class nitpick.style.StyleFetcherManager(*offline: bool, cache_dir: Path, cache_option: str*)

Bases: `object`

Manager that controls which fetcher to be used given a protocol.

cache_dir: `Path`

cache_option: `str`

fetch(*url: furl*) → `str | None`

Determine which fetcher to be used and fetch from it.

Returns None when offline is True and the fetcher would otherwise require a connection.

fetchers: `dict[str, nitpick.style.StyleFetcher]`

normalize_url(*url: str | furl, base: furl*) → `furl`

Normalize a style URL.

The URL is made absolute against base, then passed to individual fetchers to produce a canonical version of the URL.

offline: `bool`

schemes: `tuple[str]`

session: `CachedSession`

class nitpick.style.StyleManager(*project: Project, offline: bool, cache_option: str*)

Bases: `object`

Include styles recursively from one another.

property cache_dir: `Path`

Clear the cache directory (on the project root or on the current directory).

cache_option: `str`

static file_field_pair(*filename: str, base_file_class: type[NitpickPlugin]*) → `dict[str, fields.Field]`

Return a schema field with info from a config file class.

find_initial_styles(*configured_styles: Sequence[str], base: str | None = None*) → `Iterator[Fuss]`

Find the initial style(s) and include them.

base is the URL for the source of the initial styles, and is used to resolve relative references. If omitted, defaults to the project root.

static get_default_style_url(*github=False*) → `furl`

Return the URL of the default style/preset.

include_multiple_styles(*chosen_styles: Iterable[furl]*) → `Iterator[Fuss]`

Include a list of styles (or just one) into this style tree.

load_fixed_name_plugins() → `set[type[NitpickPlugin]]`

Separate classes with fixed file names from classes with dynamic file names.

merge_toml_dict() → `JsonDict`

Merge all included styles into a TOML (actually JSON) dictionary.

offline: `bool`

project: *Project*

rebuild_dynamic_schema() → None

Rebuild the dynamic Marshmallow schema when needed, adding new fields that were found on the style.

nitpick.style.builtin_resources_root() → Path

Built-in resources root.

nitpick.style.builtin_styles() → Iterable[Path]

List the built-in styles.

nitpick.style.github_default_branch(api_url: str, *, token: str | None = None) → str

Get the default branch from the GitHub repo using the API.

For now, for URLs without an authorization token embedded, the request is not authenticated on GitHub, so it might hit a rate limit with: `requests.exceptions.HTTPError: 403 Client Error: rate limit exceeded for url`

This function is using `lru_cache()` as a simple memoizer, trying to avoid this rate limit error.

nitpick.style.parse_cache_option(cache_option: str) → tuple[nitpick.constants.CachingEnum, datetime.timedelta | int]

Parse the cache option provided on `pyproject.toml`.

If no cache is provided or is invalid, the default is *one hour*.

nitpick.style.repo_root() → Path

Repository root, 3 levels up from the resources root.

nitpick.tomlkit_ext module

Extensions for the `tomlkit` package, with drop-in replacement functions and classes.

Eventually, some of the code here should/could be proposed as pull requests to the original package.

nitpick.tomlkit_ext.load(file_pointer: IO[str] | IO[bytes] | Path) → TOMLDocument

Load a TOML file from a file-like object or path.

Return an empty document if the file doesn't exist.

Drop-in replacement for `tomlkit.api.load()`.

nitpick.tomlkit_ext.multiline_comment_with_markers(marker: str, text: str) → list[str]

Add markers to a comment with multiple lines.

nitpick.tomlkit_ext.update_comment_before(table: Table, key: str, marker: str, comment: str) → None

Update a comment before a key, between markers.

Either replace an existing block or create a new one.

nitpick.typedefs module

Type definitions.

nitpick.violations module

Violation codes.

Name inspired by `flake8`'s violations.

```
class nitpick.violations.Fuss(fixed: bool, filename: str, code: int, message: str, suggestion: str = "", lineno: int = 1)
```

Bases: `object`

Nitpick makes a fuss when configuration doesn't match.

Fields inspired on `SyntaxError` and `pyflakes.messages.Message`.

code: `int`

property colored_suggestion: `str`

Suggestion with color.

filename: `str`

fixed: `bool`

lineno: `int = 1`

message: `str`

property pretty: `str`

Message to be used on the CLI.

suggestion: `str = ''`

```
class nitpick.violations.ProjectViolations(value, names=None, *, module=None, qualname=None, type=None, start=1, boundary=None)
```

Bases: `ViolationEnum`

Project initialization violations.

```
FILE_SHOULD_BE_DELETED = (104, ' should be deleted{extra}')
```

```
MINIMUM_VERSION = (203, "The style file you're using requires {project}>={expected}  
(you have {actual}). Please upgrade")
```

```
MISSING_FILE = (103, ' should exist{extra}')
```

```
NO_PYTHON_FILE = (102, 'No Python file was found on the root dir and subdir of  
{root!r}')
```

```
NO_ROOT_DIR = (101, "No root directory detected. Run 'nitpick init' or configure a  
style manually (.nitpick.toml, pyproject.toml). See  
https://nitpick.rtf.d.io/en/latest/configuration.html")
```

class nitpick.violations.**Reporter**(*info: FileInfo | None = None, violation_base_code: int = 0*)

Bases: `object`

Error reporter.

fixed: `int = 0`

classmethod `get_counts()` → `str`

String representation with error counts and emojis.

classmethod `increment(fixed=False)`

Increment the fixed ou manual count.

make_fuss(*violation: ViolationEnum, suggestion: str = "", fixed=False, **kwargs*) → `Fuss`

Make a fuss.

manual: `int = 0`

classmethod `reset()`

Reset the counters.

class nitpick.violations.**SharedViolations**(*value, names=None, *, module=None, qualname=None, type=None, start=1, boundary=None*)

Bases: `ViolationEnum`

Shared violations used by all plugins.

CREATE_FILE = (1, ' was not found', True)

CREATE_FILE_WITH_SUGGESTION = (1, ' was not found. Create it with this content:', True)

DELETE_FILE = (2, ' should be deleted', True)

DIFFERENT_VALUES = (9, '{prefix} has different values. Use this:', True)

MISSING_VALUES = (8, '{prefix} has missing values:', True)

class nitpick.violations.**StyleViolations**(*value, names=None, *, module=None, qualname=None, type=None, start=1, boundary=None*)

Bases: `ViolationEnum`

Style violations.

INVALID_CONFIG = (1, ' has an incorrect style. Invalid config:')

INVALID_DATA_TOOL_NITPICK = (1, ' has an incorrect style. Invalid data in [{section}]:')

INVALID_TOML = (1, ' has an incorrect style. Invalid TOML{exception}')

NO_STYLE_CONFIGURED = (4, "No style file configured. Run 'nitpick init' or configure a style manually (.nitpick.toml, pyproject.toml). See <https://nitpick.rtfid.io/en/latest/configuration.html>")

class nitpick.violations.**ViolationEnum**(*value, names=None, *, module=None, qualname=None, type=None, start=1, boundary=None*)

Bases: `Enum`

Base enum with violation codes and messages.

INDICES AND TABLES

- [genindex](#)
- [modindex](#)
- [search](#)

CHAPTER
FOURTEEN

TO DO LIST

PYTHON MODULE INDEX

n

- nitpick, 37
- nitpick.blender, 61
- nitpick.cli, 67
- nitpick.compat, 67
- nitpick.config, 68
- nitpick.constants, 68
- nitpick.core, 70
- nitpick.exceptions, 76
- nitpick.fields, 77
- nitpick.flake8, 85
- nitpick.generic, 86
- nitpick.plugins, 38
- nitpick.plugins.base, 38
- nitpick.plugins.info, 39
- nitpick.plugins.ini, 40
- nitpick.plugins.json, 42
- nitpick.plugins.text, 47
- nitpick.plugins.toml, 57
- nitpick.plugins.yaml, 58
- nitpick.resources, 59
- nitpick.resources.any, 60
- nitpick.resources.javascript, 60
- nitpick.resources.kotlin, 60
- nitpick.resources.markdown, 60
- nitpick.resources.presets, 60
- nitpick.resources.proto, 60
- nitpick.resources.python, 60
- nitpick.resources.shell, 60
- nitpick.resources.toml, 60
- nitpick.schemas, 86
- nitpick.style, 114
- nitpick.tomlkit_ext, 125
- nitpick.typedefs, 126
- nitpick.violations, 126

A

add_note() (*nitpick.exceptions.QuitComplainingError* method), 76
 add_options() (*nitpick.flake8.NitpickFlake8Extension* static method), 85
 add_options_before_space() (*nitpick.plugins.ini.IniPlugin* method), 40
 api_url (*nitpick.style.GitHubURL* property), 116
 args (*nitpick.exceptions.QuitComplainingError* attribute), 76
 as_integer_ratio() (*nitpick.constants.CachingEnum* method), 68
 as_object (*nitpick.blender.BaseDoc* property), 61
 as_object (*nitpick.blender.JsonDoc* property), 62
 as_object (*nitpick.blender.TomlDoc* property), 65
 as_object (*nitpick.blender.YamlDoc* property), 65
 as_string (*nitpick.blender.BaseDoc* property), 61
 as_string (*nitpick.blender.JsonDoc* property), 62
 as_string (*nitpick.blender.TomlDoc* property), 65
 as_string (*nitpick.blender.YamlDoc* property), 65
 auth_token (*nitpick.style.GitHubURL* attribute), 116
 authorization_header (*nitpick.style.GitHubURL* property), 116

B

BaseDoc (class in *nitpick.blender*), 61
 BaseNitpickSchema (class in *nitpick.schemas*), 86
 BaseNitpickSchema.Meta (class in *nitpick.schemas*), 86
 BaseStyleSchema (class in *nitpick.schemas*), 90
 BaseStyleSchema.Meta (class in *nitpick.schemas*), 90
 bit_count() (*nitpick.constants.CachingEnum* method), 68
 bit_length() (*nitpick.constants.CachingEnum* method), 69
 block_seq_indent (*nitpick.blender.SensibleYAML* property), 63
 bounded_string() (*nitpick.blender.InlineTableTomlDecoder* method), 62
 build_flake8_error() (*nitpick.flake8.NitpickFlake8Extension* method),

85

builtin_resources_root() (in module *nitpick.style*), 125
 builtin_styles() (in module *nitpick.style*), 125
 BuiltinStyle (class in *nitpick.style*), 114

C

cache (*nitpick.core.Configuration* attribute), 70
 cache_dir (*nitpick.style.StyleFetcherManager* attribute), 124
 cache_dir (*nitpick.style.StyleManager* property), 124
 cache_option (*nitpick.style.StyleFetcherManager* attribute), 124
 cache_option (*nitpick.style.StyleManager* attribute), 124
 CachingEnum (class in *nitpick.constants*), 68
 can_handle() (in module *nitpick.plugins.ini*), 42
 can_handle() (in module *nitpick.plugins.json*), 47
 can_handle() (in module *nitpick.plugins.text*), 56
 can_handle() (in module *nitpick.plugins.toml*), 58
 can_handle() (in module *nitpick.plugins.yaml*), 59
 capitalize() (*nitpick.style.Scheme* method), 118
 casefold() (*nitpick.style.Scheme* method), 118
 cast_to_dict (*nitpick.blender.ElementDetail* property), 61
 center() (*nitpick.style.Scheme* method), 118
 code (*nitpick.violations.Fuss* attribute), 126
 collect_errors() (*nitpick.flake8.NitpickFlake8Extension* method), 85
 colored_suggestion (*nitpick.violations.Fuss* property), 126
 comma_separated_values (*nitpick.plugins.ini.IniPlugin* attribute), 40
 common_fix_or_check() (in module *nitpick.cli*), 67
 compact (*nitpick.blender.ElementDetail* attribute), 61
 compact() (*nitpick.blender.SensibleYAML* method), 63
 compare_different_keys() (*nitpick.plugins.ini.IniPlugin* method), 40
 compare_lists_with_dictdiffer() (in module *nitpick.blender*), 65
 Comparison (class in *nitpick.blender*), 61

- compose() (*nitpick.blender.SensibleYAML method*), 63
- compose_all() (*nitpick.blender.SensibleYAML method*), 63
- composer (*nitpick.blender.SensibleYAML property*), 63
- config_file() (*nitpick.core.Project method*), 72
- config_file_or_default() (*nitpick.core.Project method*), 72
- Configuration (*class in nitpick.core*), 70
- configured_files() (*nitpick.core.Nitpick method*), 71
- configured_files() (*nitpick.Nitpick method*), 37
- ConfigValidator (*class in nitpick.style*), 114
- confirm_project_root() (*in module nitpick.core*), 76
- conjugate() (*nitpick.constants.CachingEnum method*), 69
- CONSTRUCTION (*nitpick.constants.EmojiEnum attribute*), 70
- constructor (*nitpick.blender.SensibleYAML property*), 63
- content_path (*nitpick.style.PythonPackageURL property*), 118
- contents_without_top_section() (*nitpick.plugins.ini.IniPlugin static method*), 40
- context (*nitpick.fields.Dict property*), 77
- context (*nitpick.fields.Field property*), 79
- context (*nitpick.fields.List property*), 81
- context (*nitpick.fields.Nested property*), 83
- context (*nitpick.fields.String property*), 84
- count() (*nitpick.style.Scheme method*), 118
- create() (*nitpick.plugins.info.FileInfo class method*), 39
- CREATE_FILE (*nitpick.violations.SharedViolations attribute*), 127
- CREATE_FILE_WITH_SUGGESTION (*nitpick.violations.SharedViolations attribute*), 127
- current_sections (*nitpick.plugins.ini.IniPlugin property*), 40
- custom_reducer() (*in module nitpick.blender*), 65
- custom_splitter() (*in module nitpick.blender*), 65
- ## D
- data (*nitpick.blender.ElementDetail attribute*), 62
- data (*nitpick.blender.ListDetail attribute*), 63
- default (*nitpick.fields.Dict property*), 77
- default (*nitpick.fields.Field property*), 79
- default (*nitpick.fields.List property*), 81
- default (*nitpick.fields.Nested property*), 83
- default (*nitpick.fields.String property*), 84
- default_branch (*nitpick.style.GitHubURL property*), 116
- default_error_messages (*nitpick.fields.Dict attribute*), 77
- default_error_messages (*nitpick.fields.Field attribute*), 79
- default_error_messages (*nitpick.fields.List attribute*), 81
- default_error_messages (*nitpick.fields.Nested attribute*), 83
- default_error_messages (*nitpick.fields.String attribute*), 84
- DELETE_FILE (*nitpick.violations.SharedViolations attribute*), 127
- denominator (*nitpick.constants.CachingEnum attribute*), 69
- Deprecation (*class in nitpick.exceptions*), 76
- deserialize() (*nitpick.fields.Dict method*), 77
- deserialize() (*nitpick.fields.Field method*), 79
- deserialize() (*nitpick.fields.List method*), 81
- deserialize() (*nitpick.fields.Nested method*), 83
- deserialize() (*nitpick.fields.String method*), 84
- Dict (*class in nitpick.fields*), 77
- dict_class (*nitpick.core.ToolNitpickSectionSchema property*), 73
- dict_class (*nitpick.plugins.json.JsonFileSchema property*), 43
- dict_class (*nitpick.plugins.text.TextItemSchema property*), 48
- dict_class (*nitpick.plugins.text.TextSchema property*), 53
- dict_class (*nitpick.schemas.BaseNitpickSchema property*), 87
- dict_class (*nitpick.schemas.BaseStyleSchema property*), 91
- dict_class (*nitpick.schemas.IniSchema property*), 95
- dict_class (*nitpick.schemas.NitpickFilesSectionSchema property*), 99
- dict_class (*nitpick.schemas.NitpickMetaSchema property*), 103
- dict_class (*nitpick.schemas.NitpickSectionSchema property*), 107
- dict_class (*nitpick.schemas.NitpickStylesSectionSchema property*), 111
- diff (*nitpick.blender.Comparison property*), 61
- DIFFERENT_VALUES (*nitpick.violations.SharedViolations attribute*), 127
- dirty (*nitpick.plugins.ini.IniPlugin attribute*), 40
- dirty (*nitpick.plugins.json.JsonPlugin attribute*), 46
- dirty (*nitpick.plugins.text.TextPlugin attribute*), 51
- dirty (*nitpick.plugins.toml.TomlPlugin attribute*), 57
- dirty (*nitpick.plugins.yaml.YamlPlugin attribute*), 58
- doc (*nitpick.core.Configuration attribute*), 70
- domains (*nitpick.style.FileFetcher attribute*), 115
- domains (*nitpick.style.GitHubFetcher attribute*), 115
- domains (*nitpick.style.HttpFetcher attribute*), 117
- domains (*nitpick.style.PythonPackageFetcher attribute*), 117
- domains (*nitpick.style.StyleFetcher attribute*), 123
- dont_suggest (*nitpick.core.Configuration attribute*), 70

- dump() (*nitpick.blender.SensibleYAML method*), 63
 dump() (*nitpick.core.ToolNitpickSectionSchema method*), 73
 dump() (*nitpick.plugins.json.JsonFileSchema method*), 43
 dump() (*nitpick.plugins.text.TextItemSchema method*), 48
 dump() (*nitpick.plugins.text.TextSchema method*), 53
 dump() (*nitpick.schemas.BaseNitpickSchema method*), 87
 dump() (*nitpick.schemas.BaseStyleSchema method*), 91
 dump() (*nitpick.schemas.IniSchema method*), 95
 dump() (*nitpick.schemas.NitpickFilesSectionSchema method*), 99
 dump() (*nitpick.schemas.NitpickMetaSchema method*), 103
 dump() (*nitpick.schemas.NitpickSectionSchema method*), 107
 dump() (*nitpick.schemas.NitpickStylesSectionSchema method*), 111
 dump_all() (*nitpick.blender.SensibleYAML method*), 63
 dump_fields (*nitpick.schemas.IniSchema attribute*), 95
 dump_fields (*nitpick.schemas.NitpickFilesSectionSchema attribute*), 99
 dump_fields (*nitpick.schemas.NitpickMetaSchema attribute*), 103
 dump_fields (*nitpick.schemas.NitpickSectionSchema attribute*), 107
 dump_fields (*nitpick.schemas.NitpickStylesSectionSchema attribute*), 111
 dumps() (*nitpick.blender.SensibleYAML method*), 63
 dumps() (*nitpick.core.ToolNitpickSectionSchema method*), 74
 dumps() (*nitpick.plugins.json.JsonFileSchema method*), 44
 dumps() (*nitpick.plugins.text.TextItemSchema method*), 49
 dumps() (*nitpick.plugins.text.TextSchema method*), 54
 dumps() (*nitpick.schemas.BaseNitpickSchema method*), 88
 dumps() (*nitpick.schemas.BaseStyleSchema method*), 92
 dumps() (*nitpick.schemas.IniSchema method*), 96
 dumps() (*nitpick.schemas.NitpickFilesSectionSchema method*), 99
 dumps() (*nitpick.schemas.NitpickMetaSchema method*), 103
 dumps() (*nitpick.schemas.NitpickSectionSchema method*), 107
 dumps() (*nitpick.schemas.NitpickStylesSectionSchema method*), 111
- ## E
- echo() (*nitpick.core.Nitpick method*), 71
 echo() (*nitpick.Nitpick method*), 37
 ElementDetail (*class in nitpick.blender*), 61
 elements (*nitpick.blender.ListDetail attribute*), 63
 embed_comments() (*nitpick.blender.InlineTableTomlDecoder method*), 62
 emit() (*nitpick.blender.SensibleYAML method*), 63
 emitter (*nitpick.blender.SensibleYAML property*), 63
 EmojiEnum (*class in nitpick.constants*), 70
 encode() (*nitpick.style.Scheme method*), 119
 endswith() (*nitpick.style.Scheme method*), 119
 enforce_comma_separated_values() (*nitpick.plugins.ini.IniPlugin method*), 40
 enforce_missing_sections() (*nitpick.plugins.ini.IniPlugin method*), 40
 enforce_present_absent() (*nitpick.core.Nitpick method*), 71
 enforce_present_absent() (*nitpick.Nitpick method*), 37
 enforce_rules() (*nitpick.plugins.base.NitpickPlugin method*), 38
 enforce_rules() (*nitpick.plugins.ini.IniPlugin method*), 40
 enforce_rules() (*nitpick.plugins.json.JsonPlugin method*), 46
 enforce_rules() (*nitpick.plugins.text.TextPlugin method*), 51
 enforce_rules() (*nitpick.plugins.toml.TomlPlugin method*), 57
 enforce_rules() (*nitpick.plugins.yaml.YamlPlugin method*), 58
 enforce_section() (*nitpick.plugins.ini.IniPlugin method*), 40
 enforce_style() (*nitpick.core.Nitpick method*), 71
 enforce_style() (*nitpick.Nitpick method*), 37
 entry_point() (*nitpick.plugins.base.NitpickPlugin method*), 38
 entry_point() (*nitpick.plugins.ini.IniPlugin method*), 40
 entry_point() (*nitpick.plugins.json.JsonPlugin method*), 46
 entry_point() (*nitpick.plugins.text.TextPlugin method*), 51
 entry_point() (*nitpick.plugins.toml.TomlPlugin method*), 57
 entry_point() (*nitpick.plugins.yaml.YamlPlugin method*), 58
 error_messages (*nitpick.core.ToolNitpickSectionSchema attribute*), 74
 error_messages (*nitpick.plugins.json.JsonFileSchema attribute*), 44
 error_messages (*nitpick.plugins.text.TextItemSchema attribute*), 49
 error_messages (*nitpick.plugins.text.TextSchema attribute*), 54
 error_messages (*nitpick.schemas.BaseNitpickSchema*

- `attribute`), 88
 - `error_messages` (*nitpick.schemas.BaseStyleSchema attribute*), 92
 - `error_messages` (*nitpick.schemas.IniSchema attribute*), 96
 - `error_messages` (*nitpick.schemas.NitpickFilesSectionSchema attribute*), 100
 - `error_messages` (*nitpick.schemas.NitpickMetaSchema attribute*), 104
 - `error_messages` (*nitpick.schemas.NitpickSectionSchema attribute*), 108
 - `error_messages` (*nitpick.schemas.NitpickStylesSectionSchema attribute*), 112
 - `exclude` (*nitpick.schemas.IniSchema attribute*), 96
 - `exclude` (*nitpick.schemas.NitpickFilesSectionSchema attribute*), 100
 - `exclude` (*nitpick.schemas.NitpickMetaSchema attribute*), 104
 - `exclude` (*nitpick.schemas.NitpickSectionSchema attribute*), 108
 - `exclude` (*nitpick.schemas.NitpickStylesSectionSchema attribute*), 112
 - `expandtabs()` (*nitpick.style.Scheme method*), 119
 - `expected_config` (*nitpick.plugins.ini.IniPlugin attribute*), 40
 - `expected_config` (*nitpick.plugins.json.JsonPlugin attribute*), 46
 - `expected_config` (*nitpick.plugins.text.TextPlugin attribute*), 51
 - `expected_config` (*nitpick.plugins.toml.TomlPlugin attribute*), 57
 - `expected_config` (*nitpick.plugins.yaml.YamlPlugin attribute*), 58
 - `expected_dict_from_contains_json()` (*nitpick.plugins.json.JsonPlugin method*), 46
 - `expected_dict_from_contains_keys()` (*nitpick.plugins.json.JsonPlugin method*), 46
 - `expected_sections` (*nitpick.plugins.ini.IniPlugin property*), 40
 - `EXPIRES` (*nitpick.constants.CachingEnum attribute*), 68
- ## F
- `fail()` (*nitpick.fields.Dict method*), 77
 - `fail()` (*nitpick.fields.Field method*), 80
 - `fail()` (*nitpick.fields.List method*), 81
 - `fail()` (*nitpick.fields.Nested method*), 83
 - `fail()` (*nitpick.fields.String method*), 84
 - `fetch()` (*nitpick.style.FileFetcher method*), 115
 - `fetch()` (*nitpick.style.GitHubFetcher method*), 115
 - `fetch()` (*nitpick.style.HttpFetcher method*), 117
 - `fetch()` (*nitpick.style.PythonPackageFetcher method*), 117
 - `fetch()` (*nitpick.style.StyleFetcher method*), 123
 - `fetch()` (*nitpick.style.StyleFetcherManager method*), 124
 - `fetchers` (*nitpick.style.StyleFetcherManager attribute*), 124
 - `Field` (*class in nitpick.fields*), 78
 - `fields` (*nitpick.core.ToolNitpickSectionSchema attribute*), 74
 - `fields` (*nitpick.plugins.json.JsonFileSchema attribute*), 44
 - `fields` (*nitpick.plugins.text.TextItemSchema attribute*), 49
 - `fields` (*nitpick.plugins.text.TextSchema attribute*), 54
 - `fields` (*nitpick.schemas.BaseNitpickSchema attribute*), 88
 - `fields` (*nitpick.schemas.BaseStyleSchema attribute*), 92
 - `fields` (*nitpick.schemas.IniSchema attribute*), 96
 - `fields` (*nitpick.schemas.NitpickFilesSectionSchema attribute*), 100
 - `fields` (*nitpick.schemas.NitpickMetaSchema attribute*), 104
 - `fields` (*nitpick.schemas.NitpickSectionSchema attribute*), 108
 - `fields` (*nitpick.schemas.NitpickStylesSectionSchema attribute*), 112
 - `file` (*nitpick.core.Configuration attribute*), 70
 - `FILE` (*nitpick.style.Scheme attribute*), 118
 - `file_field_pair()` (*nitpick.style.StyleManager static method*), 124
 - `file_path` (*nitpick.plugins.ini.IniPlugin attribute*), 40
 - `file_path` (*nitpick.plugins.json.JsonPlugin attribute*), 46
 - `file_path` (*nitpick.plugins.text.TextPlugin attribute*), 51
 - `file_path` (*nitpick.plugins.toml.TomlPlugin attribute*), 57
 - `file_path` (*nitpick.plugins.yaml.YamlPlugin attribute*), 58
 - `FILE_SHOULD_BE_DELETED` (*nitpick.violations.ProjectViolations attribute*), 126
 - `FileFetcher` (*class in nitpick.style*), 114
 - `FileInfo` (*class in nitpick.plugins.info*), 39
 - `filename` (*nitpick.plugins.base.NitpickPlugin attribute*), 38
 - `filename` (*nitpick.plugins.ini.IniPlugin attribute*), 40
 - `filename` (*nitpick.plugins.json.JsonPlugin attribute*), 46
 - `filename` (*nitpick.plugins.text.TextPlugin attribute*), 51
 - `filename` (*nitpick.plugins.toml.TomlPlugin attribute*), 57
 - `filename` (*nitpick.plugins.yaml.YamlPlugin attribute*), 58
 - `filename` (*nitpick.violations.Fuss attribute*), 126
 - `files` (*nitpick.style.BuiltinStyle attribute*), 114
 - `filter_names()` (*in module nitpick.generic*), 86
 - `find()` (*nitpick.style.Scheme method*), 119
 - `find_by_key()` (*nitpick.blender.ListDetail method*), 63

- `find_initial_styles()` (*nitpick.style.StyleManager* method), 124
`find_main_python_file()` (in module *nitpick.core*), 76
`fixable` (*nitpick.plugins.base.NitpickPlugin* attribute), 38
`fixable` (*nitpick.plugins.ini.IniPlugin* attribute), 41
`fixable` (*nitpick.plugins.json.JsonPlugin* attribute), 46
`fixable` (*nitpick.plugins.text.TextPlugin* attribute), 52
`fixable` (*nitpick.plugins.toml.TomlPlugin* attribute), 57
`fixable` (*nitpick.plugins.yaml.YamlPlugin* attribute), 58
`fixed` (*nitpick.violations.Fuss* attribute), 126
`fixed` (*nitpick.violations.Reporter* attribute), 127
`Flake8OptionEnum` (class in *nitpick.constants*), 70
`flatten_marshalmallow_errors()` (in module *nitpick.schemas*), 114
`flatten_quotes()` (in module *nitpick.blender*), 65
`FOREVER` (*nitpick.constants.CachingEnum* attribute), 68
`format()` (*nitpick.style.Scheme* method), 119
`format_map()` (*nitpick.style.Scheme* method), 119
`formatted` (*nitpick.style.BuiltinStyle* attribute), 114
`from_bytes()` (*nitpick.constants.CachingEnum* method), 69
`from_data()` (*nitpick.blender.ElementDetail* class method), 62
`from_data()` (*nitpick.blender.ListDetail* class method), 63
`from_dict()` (*nitpick.core.ToolNitpickSectionSchema* class method), 74
`from_dict()` (*nitpick.plugins.json.JsonFileSchema* class method), 44
`from_dict()` (*nitpick.plugins.text.TextItemSchema* class method), 49
`from_dict()` (*nitpick.plugins.text.TextSchema* class method), 54
`from_dict()` (*nitpick.schemas.BaseNitpickSchema* class method), 88
`from_dict()` (*nitpick.schemas.BaseStyleSchema* class method), 92
`from_dict()` (*nitpick.schemas.IniSchema* class method), 96
`from_dict()` (*nitpick.schemas.NitpickFilesSectionSchema* class method), 100
`from_dict()` (*nitpick.schemas.NitpickMetaSchema* class method), 104
`from_dict()` (*nitpick.schemas.NitpickSectionSchema* class method), 108
`from_dict()` (*nitpick.schemas.NitpickStylesSectionSchema* class method), 112
`from_furl()` (*nitpick.style.GitHubURL* class method), 116
`from_furl()` (*nitpick.style.PythonPackageURL* class method), 118
`from_path()` (*nitpick.style.BuiltinStyle* class method), 114
`from_plugin` (*nitpick.config.OverridableConfig* attribute), 68
`from_style` (*nitpick.config.OverridableConfig* attribute), 68
`Fuss` (class in *nitpick.violations*), 126
- ## G
- `get_attribute()` (*nitpick.core.ToolNitpickSectionSchema* method), 74
`get_attribute()` (*nitpick.plugins.json.JsonFileSchema* method), 44
`get_attribute()` (*nitpick.plugins.text.TextItemSchema* method), 49
`get_attribute()` (*nitpick.plugins.text.TextSchema* method), 54
`get_attribute()` (*nitpick.schemas.BaseNitpickSchema* method), 88
`get_attribute()` (*nitpick.schemas.BaseStyleSchema* method), 92
`get_attribute()` (*nitpick.schemas.IniSchema* method), 96
`get_attribute()` (*nitpick.schemas.NitpickFilesSectionSchema* method), 100
`get_attribute()` (*nitpick.schemas.NitpickMetaSchema* method), 104
`get_attribute()` (*nitpick.schemas.NitpickSectionSchema* method), 108
`get_attribute()` (*nitpick.schemas.NitpickStylesSectionSchema* method), 112
`get_constructor_parser()` (*nitpick.blender.SensibleYAML* method), 64
`get_counts()` (*nitpick.violations.Reporter* class method), 127
`get_default_style_url()` (*nitpick.style.StyleManager* static method), 124
`get_empty_inline_table()` (*nitpick.blender.InlineTableTomlDecoder* method), 62
`get_empty_table()` (*nitpick.blender.InlineTableTomlDecoder* method), 62
`get_example_cfg()` (*nitpick.plugins.ini.IniPlugin* static method), 41
`get_global_gitignore_path()` (in module *nitpick.generic*), 86
`get_missing_output()` (*nitpick.plugins.ini.IniPlugin* method), 41
`get_nitpick()` (in module *nitpick.cli*), 67

- get_serializer_representer_emitter() (*nitpick.blender.SensibleYAML method*), 64
 get_value() (*nitpick.fields.Dict method*), 77
 get_value() (*nitpick.fields.Field method*), 80
 get_value() (*nitpick.fields.List method*), 81
 get_value() (*nitpick.fields.Nested method*), 83
 get_value() (*nitpick.fields.String method*), 84
 GH (*nitpick.style.Scheme attribute*), 118
 git_reference (*nitpick.style.GitHubURL attribute*), 116
 git_reference_or_default (*nitpick.style.GitHubURL property*), 116
 GITHUB (*nitpick.style.Scheme attribute*), 118
 github_default_branch() (*in module nitpick.style*), 125
 GitHubFetcher (*class in nitpick.style*), 115
 GitHubURL (*class in nitpick.style*), 116
 glob_files() (*in module nitpick.generic*), 86
 glob_non_ignored_files() (*in module nitpick.generic*), 86
 GREEN_CHECK (*nitpick.constants.EmojiEnum attribute*), 70
- ## H
- handle_error() (*nitpick.core.ToolNitpickSectionSchema method*), 74
 handle_error() (*nitpick.plugins.json.JsonFileSchema method*), 44
 handle_error() (*nitpick.plugins.text.TextItemSchema method*), 50
 handle_error() (*nitpick.plugins.text.TextSchema method*), 55
 handle_error() (*nitpick.schemas.BaseNitpickSchema method*), 89
 handle_error() (*nitpick.schemas.BaseStyleSchema method*), 92
 handle_error() (*nitpick.schemas.IniSchema method*), 96
 handle_error() (*nitpick.schemas.NitpickFilesSectionSchema method*), 100
 handle_error() (*nitpick.schemas.NitpickMetaSchema method*), 104
 handle_error() (*nitpick.schemas.NitpickSectionSchema method*), 108
 handle_error() (*nitpick.schemas.NitpickStylesSectionSchema method*), 112
 has_changes (*nitpick.blender.Comparison property*), 61
 help_message() (*in module nitpick.schemas*), 114
 HTTP (*nitpick.style.Scheme attribute*), 118
 HttpFetcher (*class in nitpick.style*), 116
 HTTPS (*nitpick.style.Scheme attribute*), 118
- ## I
- identify_tag (*nitpick.style.BuiltinStyle attribute*), 114
 identify_tags (*nitpick.plugins.base.NitpickPlugin attribute*), 38
 identify_tags (*nitpick.plugins.ini.IniPlugin attribute*), 41
 identify_tags (*nitpick.plugins.json.JsonPlugin attribute*), 46
 identify_tags (*nitpick.plugins.text.TextPlugin attribute*), 52
 identify_tags (*nitpick.plugins.toml.TomlPlugin attribute*), 57
 identify_tags (*nitpick.plugins.yaml.YamlPlugin attribute*), 59
 imag (*nitpick.constants.CachingEnum attribute*), 69
 import_path (*nitpick.style.PythonPackageURL attribute*), 118
 include_multiple_styles() (*nitpick.style.StyleManager method*), 124
 increment() (*nitpick.violations.Reporter class method*), 127
 indent (*nitpick.blender.SensibleYAML property*), 64
 index (*nitpick.blender.ElementDetail attribute*), 62
 index() (*nitpick.style.Scheme method*), 119
 IniPlugin (*class in nitpick.plugins.ini*), 40
 IniSchema (*class in nitpick.schemas*), 94
 IniSchema.Meta (*class in nitpick.schemas*), 94
 init() (*nitpick.core.Nitpick method*), 71
 init() (*nitpick.Nitpick method*), 37
 initial_contents (*nitpick.plugins.base.NitpickPlugin property*), 39
 initial_contents (*nitpick.plugins.ini.IniPlugin property*), 41
 initial_contents (*nitpick.plugins.json.JsonPlugin property*), 46
 initial_contents (*nitpick.plugins.text.TextPlugin property*), 52
 initial_contents (*nitpick.plugins.toml.TomlPlugin property*), 57
 initial_contents (*nitpick.plugins.yaml.YamlPlugin property*), 59
 InlineTableTomlDecoder (*class in nitpick.blender*), 62
 INVALID_COMMA_SEPARATED_VALUES_SECTION (*nitpick.plugins.ini.Violations attribute*), 41
 INVALID_CONFIG (*nitpick.violations.StyleViolations attribute*), 127
 INVALID_DATA_TOOL_NITPICK (*nitpick.violations.StyleViolations attribute*), 127
 INVALID_TOML (*nitpick.violations.StyleViolations attribute*), 127
 is_scalar() (*in module nitpick.blender*), 65
 isalnum() (*nitpick.style.Scheme method*), 119
 isalpha() (*nitpick.style.Scheme method*), 119
 isascii() (*nitpick.style.Scheme method*), 119

isdecimal() (*nitpick.style.Scheme method*), 119
 isdigit() (*nitpick.style.Scheme method*), 120
 isidentifier() (*nitpick.style.Scheme method*), 120
 islower() (*nitpick.style.Scheme method*), 120
 isnumeric() (*nitpick.style.Scheme method*), 120
 isprintable() (*nitpick.style.Scheme method*), 120
 isspace() (*nitpick.style.Scheme method*), 120
 istitle() (*nitpick.style.Scheme method*), 120
 isupper() (*nitpick.style.Scheme method*), 120

J

join() (*nitpick.style.Scheme method*), 120
 JsonDoc (*class in nitpick.blender*), 62
 jsonfile_section() (*nitpick.exceptions.Deprecation static method*), 76
 JsonFileSchema (*class in nitpick.plugins.json*), 42
 JsonFileSchema.Meta (*class in nitpick.plugins.json*), 42
 JsonPlugin (*class in nitpick.plugins.json*), 46

K

key (*nitpick.blender.ElementDetail attribute*), 62

L

lineno (*nitpick.violations.Fuss attribute*), 126
 List (*class in nitpick.fields*), 80
 list_keys (*nitpick.config.SpecialConfig attribute*), 68
 ListDetail (*class in nitpick.blender*), 62
 ljust() (*nitpick.style.Scheme method*), 120
 load() (*in module nitpick.tomlkit_ext*), 125
 load() (*nitpick.blender.BaseDoc method*), 61
 load() (*nitpick.blender.JsonDoc method*), 62
 load() (*nitpick.blender.SensibleYAML method*), 64
 load() (*nitpick.blender.TomlDoc method*), 65
 load() (*nitpick.blender.YamlDoc method*), 65
 load() (*nitpick.core.ToolNitpickSectionSchema method*), 75
 load() (*nitpick.plugins.json.JsonFileSchema method*), 45
 load() (*nitpick.plugins.text.TextItemSchema method*), 50
 load() (*nitpick.plugins.text.TextSchema method*), 55
 load() (*nitpick.schemas.BaseNitpickSchema method*), 89
 load() (*nitpick.schemas.BaseStyleSchema method*), 93
 load() (*nitpick.schemas.IniSchema method*), 97
 load() (*nitpick.schemas.NitpickFilesSectionSchema method*), 101
 load() (*nitpick.schemas.NitpickMetaSchema method*), 104
 load() (*nitpick.schemas.NitpickSectionSchema method*), 108
 load() (*nitpick.schemas.NitpickStylesSectionSchema method*), 112

load_all() (*nitpick.blender.SensibleYAML method*), 64
 load_array() (*nitpick.blender.InlineTableTomlDecoder method*), 62
 load_fields (*nitpick.schemas.IniSchema attribute*), 97
 load_fields (*nitpick.schemas.NitpickFilesSectionSchema attribute*), 101
 load_fields (*nitpick.schemas.NitpickMetaSchema attribute*), 105
 load_fields (*nitpick.schemas.NitpickSectionSchema attribute*), 109
 load_fields (*nitpick.schemas.NitpickStylesSectionSchema attribute*), 113
 load_fixed_name_plugins() (*nitpick.style.StyleManager method*), 124
 load_inline_object() (*nitpick.blender.InlineTableTomlDecoder method*), 62
 load_line() (*nitpick.blender.InlineTableTomlDecoder method*), 62
 load_value() (*nitpick.blender.InlineTableTomlDecoder method*), 62
 loads() (*nitpick.blender.SensibleYAML method*), 64
 loads() (*nitpick.core.ToolNitpickSectionSchema method*), 75
 loads() (*nitpick.plugins.json.JsonFileSchema method*), 45
 loads() (*nitpick.plugins.text.TextItemSchema method*), 50
 loads() (*nitpick.plugins.text.TextSchema method*), 55
 loads() (*nitpick.schemas.BaseNitpickSchema method*), 89
 loads() (*nitpick.schemas.BaseStyleSchema method*), 93
 loads() (*nitpick.schemas.IniSchema method*), 97
 loads() (*nitpick.schemas.NitpickFilesSectionSchema method*), 101
 loads() (*nitpick.schemas.NitpickMetaSchema method*), 105
 loads() (*nitpick.schemas.NitpickSectionSchema method*), 109
 loads() (*nitpick.schemas.NitpickStylesSectionSchema method*), 113
 long_protocol_url (*nitpick.style.GitHubURL property*), 116
 lower() (*nitpick.style.Scheme method*), 121
 lstrip() (*nitpick.style.Scheme method*), 121

M

make_error() (*nitpick.fields.Dict method*), 78
 make_error() (*nitpick.fields.Field method*), 80
 make_error() (*nitpick.fields.List method*), 81
 make_error() (*nitpick.fields.Nested method*), 83
 make_error() (*nitpick.fields.String method*), 85
 make_fuss() (*nitpick.violations.Reporter method*), 127
 maketrans() (*nitpick.style.Scheme static method*), 121

- manual (*nitpick.violations.Reporter attribute*), 127
 map() (*nitpick.blender.SensibleYAML method*), 64
 mapping_type (*nitpick.fields.Dict attribute*), 78
 merge_styles() (*nitpick.core.Project method*), 72
 merge_toml_dict() (*nitpick.style.StyleManager method*), 124
 message (*nitpick.violations.Fuss attribute*), 126
 MINIMUM_VERSION (*nitpick.violations.ProjectViolations attribute*), 126
 missing (*nitpick.blender.Comparison property*), 61
 missing (*nitpick.fields.Dict property*), 78
 missing (*nitpick.fields.Field property*), 80
 missing (*nitpick.fields.List property*), 81
 missing (*nitpick.fields.Nested property*), 83
 missing (*nitpick.fields.String property*), 85
 MISSING_FILE (*nitpick.violations.ProjectViolations attribute*), 126
 MISSING_LINES (*nitpick.plugins.text.Violations attribute*), 56
 MISSING_OPTION (*nitpick.plugins.ini.Violations attribute*), 42
 missing_sections (*nitpick.plugins.ini.IniPlugin property*), 41
 MISSING_SECTIONS (*nitpick.plugins.ini.Violations attribute*), 42
 MISSING_VALUES (*nitpick.violations.SharedViolations attribute*), 127
 MISSING_VALUES_IN_LIST (*nitpick.plugins.ini.Violations attribute*), 42
 module
 nitpick, 37
 nitpick.blender, 61
 nitpick.cli, 67
 nitpick.compat, 67
 nitpick.config, 68
 nitpick.constants, 68
 nitpick.core, 70
 nitpick.exceptions, 76
 nitpick.fields, 77
 nitpick.flake8, 85
 nitpick.generic, 86
 nitpick.plugins, 38
 nitpick.plugins.base, 38
 nitpick.plugins.info, 39
 nitpick.plugins.ini, 40
 nitpick.plugins.json, 42
 nitpick.plugins.text, 47
 nitpick.plugins.toml, 57
 nitpick.plugins.yaml, 58
 nitpick.plugins.yml, 58
 nitpick.resources, 59
 nitpick.resources.any, 60
 nitpick.resources.javascript, 60
 nitpick.resources.kotlin, 60
 nitpick.resources.markdown, 60
 nitpick.resources.presets, 60
 nitpick.resources.proto, 60
 nitpick.resources.python, 60
 nitpick.resources.shell, 60
 nitpick.resources.toml, 60
 nitpick.schemas, 86
 nitpick.style, 114
 nitpick.tomlkit_ext, 125
 nitpick.typedefs, 126
 nitpick.violations, 126
 multiline_comment_with_markers() (*in module nitpick.tomlkit_ext*), 125
- ## N
- name (*nitpick.constants.Flake8OptionEnum attribute*), 70
 name (*nitpick.fields.Dict attribute*), 78
 name (*nitpick.fields.Field attribute*), 80
 name (*nitpick.fields.List attribute*), 81
 name (*nitpick.fields.Nested attribute*), 83
 name (*nitpick.fields.String attribute*), 85
 name (*nitpick.flake8.NitpickFlake8Extension attribute*), 85
 name (*nitpick.style.BuiltinStyle attribute*), 114
 needs_top_section (*nitpick.plugins.ini.IniPlugin property*), 41
 Nested (*class in nitpick.fields*), 82
 NEVER (*nitpick.constants.CachingEnum attribute*), 68
 nitpick
 module, 37
 Nitpick (*class in nitpick*), 37
 Nitpick (*class in nitpick.core*), 70
 nitpick.blender
 module, 61
 nitpick.cli
 module, 67
 nitpick.compat
 module, 67
 nitpick.config
 module, 68
 nitpick.constants
 module, 68
 nitpick.core
 module, 70
 nitpick.exceptions
 module, 76
 nitpick.fields
 module, 77
 nitpick.flake8
 module, 85
 nitpick.generic
 module, 86
 nitpick.plugins
 module, 38
 nitpick.plugins.base

- module, 38
 - nitpick.plugins.info
 - module, 39
 - nitpick.plugins.ini
 - module, 40
 - nitpick.plugins.json
 - module, 42
 - nitpick.plugins.text
 - module, 47
 - nitpick.plugins.toml
 - module, 57
 - nitpick.plugins.yaml
 - module, 58
 - nitpick.resources
 - module, 59
 - nitpick.resources.any
 - module, 60
 - nitpick.resources.javascript
 - module, 60
 - nitpick.resources.kotlin
 - module, 60
 - nitpick.resources.markdown
 - module, 60
 - nitpick.resources.presets
 - module, 60
 - nitpick.resources.proto
 - module, 60
 - nitpick.resources.python
 - module, 60
 - nitpick.resources.shell
 - module, 60
 - nitpick.resources.toml
 - module, 60
 - nitpick.schemas
 - module, 86
 - nitpick.style
 - module, 114
 - nitpick.tomlkit_ext
 - module, 125
 - nitpick.typedefs
 - module, 126
 - nitpick.violations
 - module, 126
 - nitpick_file_dict (nitpick.plugins.base.NitpickPlugin property), 39
 - nitpick_file_dict (nitpick.plugins.ini.IniPlugin property), 41
 - nitpick_file_dict (nitpick.plugins.json.JsonPlugin property), 46
 - nitpick_file_dict (nitpick.plugins.text.TextPlugin property), 52
 - nitpick_file_dict (nitpick.plugins.toml.TomlPlugin property), 57
 - nitpick_file_dict (nitpick.plugins.yaml.YamlPlugin property), 59
 - NitpickFilesSectionSchema (class in nitpick.schemas), 98
 - NitpickFilesSectionSchema.Meta (class in nitpick.schemas), 98
 - NitpickFlake8Extension (class in nitpick.flake8), 85
 - NitpickMetaSchema (class in nitpick.schemas), 102
 - NitpickMetaSchema.Meta (class in nitpick.schemas), 102
 - NitpickPlugin (class in nitpick.plugins.base), 38
 - NitpickSectionSchema (class in nitpick.schemas), 106
 - NitpickSectionSchema.Meta (class in nitpick.schemas), 106
 - NitpickStylesSectionSchema (class in nitpick.schemas), 110
 - NitpickStylesSectionSchema.Meta (class in nitpick.schemas), 110
 - NO_PYTHON_FILE (nitpick.violations.ProjectViolations attribute), 126
 - NO_ROOT_DIR (nitpick.violations.ProjectViolations attribute), 126
 - NO_STYLE_CONFIGURED (nitpick.violations.StyleViolations attribute), 127
 - normalize() (nitpick.style.FileFetcher method), 115
 - normalize() (nitpick.style.GitHubFetcher method), 115
 - normalize() (nitpick.style.HttpFetcher method), 117
 - normalize() (nitpick.style.PythonPackageFetcher method), 117
 - normalize() (nitpick.style.StyleFetcher method), 123
 - normalize_url() (nitpick.style.StyleFetcherManager method), 124
 - numerator (nitpick.constants.CachingEnum attribute), 69
- ## O
- official_plug_ins() (nitpick.blender.SensibleYAML method), 64
 - OFFLINE (nitpick.constants.Flake8OptionEnum attribute), 70
 - offline (nitpick.core.Nitpick attribute), 71
 - offline (nitpick.style.StyleFetcherManager attribute), 124
 - offline (nitpick.style.StyleManager attribute), 124
 - on_bind_field() (nitpick.core.ToolNitpickSectionSchema method), 75
 - on_bind_field() (nitpick.plugins.json.JsonFileSchema method), 45
 - on_bind_field() (nitpick.plugins.text.TextItemSchema method), 51
 - on_bind_field() (nitpick.plugins.text.TextSchema method), 56

- `on_bind_field()` (*nitpick.schemas.BaseNitpickSchema* method), 90
`on_bind_field()` (*nitpick.schemas.BaseStyleSchema* method), 94
`on_bind_field()` (*nitpick.schemas.IniSchema* method), 97
`on_bind_field()` (*nitpick.schemas.NitpickFilesSectionSchema* method), 101
`on_bind_field()` (*nitpick.schemas.NitpickMetaSchema* method), 105
`on_bind_field()` (*nitpick.schemas.NitpickSectionSchema* method), 109
`on_bind_field()` (*nitpick.schemas.NitpickStylesSectionSchema* method), 113
OPTION_HAS_DIFFERENT_VALUE (*nitpick.plugins.ini.Violations* attribute), 42
OPTIONS_CLASS (*nitpick.core.ToolNitpickSectionSchema* attribute), 73
OPTIONS_CLASS (*nitpick.plugins.json.JsonFileSchema* attribute), 43
OPTIONS_CLASS (*nitpick.plugins.text.TextItemSchema* attribute), 48
OPTIONS_CLASS (*nitpick.plugins.text.TextSchema* attribute), 53
OPTIONS_CLASS (*nitpick.schemas.BaseNitpickSchema* attribute), 87
OPTIONS_CLASS (*nitpick.schemas.BaseStyleSchema* attribute), 91
OPTIONS_CLASS (*nitpick.schemas.IniSchema* attribute), 95
OPTIONS_CLASS (*nitpick.schemas.NitpickFilesSectionSchema* attribute), 99
OPTIONS_CLASS (*nitpick.schemas.NitpickMetaSchema* attribute), 103
OPTIONS_CLASS (*nitpick.schemas.NitpickSectionSchema* attribute), 107
OPTIONS_CLASS (*nitpick.schemas.NitpickStylesSectionSchema* attribute), 111
opts (*nitpick.core.ToolNitpickSectionSchema* attribute), 75
opts (*nitpick.plugins.json.JsonFileSchema* attribute), 45
opts (*nitpick.plugins.text.TextItemSchema* attribute), 51
opts (*nitpick.plugins.text.TextSchema* attribute), 56
opts (*nitpick.schemas.BaseNitpickSchema* attribute), 90
opts (*nitpick.schemas.BaseStyleSchema* attribute), 94
opts (*nitpick.schemas.IniSchema* attribute), 97
opts (*nitpick.schemas.NitpickFilesSectionSchema* attribute), 101
opts (*nitpick.schemas.NitpickMetaSchema* attribute), 105
opts (*nitpick.schemas.NitpickSectionSchema* attribute), 109
opts (*nitpick.schemas.NitpickStylesSectionSchema* attribute), 113
OverridableConfig (class in *nitpick.config*), 68
owner (*nitpick.style.GitHubURL* attribute), 116
- ## P
- parent** (*nitpick.fields.Dict* attribute), 78
parent (*nitpick.fields.Field* attribute), 80
parent (*nitpick.fields.List* attribute), 81
parent (*nitpick.fields.Nested* attribute), 83
parent (*nitpick.fields.String* attribute), 85
parse() (*nitpick.blender.SensibleYAML* method), 64
parse_cache_option() (in module *nitpick.style*), 125
parse_options() (*nitpick.flake8.NitpickFlake8Extension* static method), 85
parser (*nitpick.blender.SensibleYAML* property), 64
PARSING_ERROR (*nitpick.plugins.ini.Violations* attribute), 42
partition() (*nitpick.style.Scheme* method), 121
path (*nitpick.style.GitHubURL* attribute), 116
path_from_resources_root (*nitpick.style.BuiltinStyle* attribute), 114
path_from_root (*nitpick.plugins.info.FileInfo* attribute), 39
plugin_class() (in module *nitpick.plugins.ini*), 42
plugin_class() (in module *nitpick.plugins.json*), 47
plugin_class() (in module *nitpick.plugins.text*), 56
plugin_class() (in module *nitpick.plugins.toml*), 58
plugin_class() (in module *nitpick.plugins.yaml*), 59
plugin_manager (*nitpick.core.Project* property), 72
post_init() (*nitpick.plugins.base.NitpickPlugin* method), 39
post_init() (*nitpick.plugins.ini.IniPlugin* method), 41
post_init() (*nitpick.plugins.json.JsonPlugin* method), 47
post_init() (*nitpick.plugins.text.TextPlugin* method), 52
post_init() (*nitpick.plugins.toml.TomlPlugin* method), 57
post_init() (*nitpick.plugins.yaml.YamlPlugin* method), 59
pre_commit_repos_with_yaml_key() (*nitpick.exceptions.Deprecation* static method), 76
pre_commit_without_dash() (*nitpick.exceptions.Deprecation* static method), 76
predefined_special_config() (*nitpick.plugins.base.NitpickPlugin* method), 39
predefined_special_config() (*nitpick.plugins.ini.IniPlugin* method), 41

- ```
predefined_special_config()
```

 (*nitpick.plugins.json.JsonPlugin method*), 47  

```
predefined_special_config()
```

 (*nitpick.plugins.text.TextPlugin method*), 52  

```
predefined_special_config()
```

 (*nitpick.plugins.toml.TomlPlugin method*), 57  

```
predefined_special_config()
```

 (*nitpick.plugins.yaml.YamlPlugin method*), 59  

```
preprocess_relative_url()
```

 (*nitpick.style.FileFetcher method*), 115  

```
preprocess_relative_url()
```

 (*nitpick.style.GitHubFetcher method*), 115  

```
preprocess_relative_url()
```

 (*nitpick.style.HttpFetcher method*), 117  

```
preprocess_relative_url()
```

 (*nitpick.style.PythonPackageFetcher method*), 117  

```
preprocess_relative_url()
```

 (*nitpick.style.StyleFetcher method*), 123  

```
preserve_comment()
```

 (*nitpick.blender.InlineTableTomlDecoder method*), 62  

```
pretty
```

 (*nitpick.violations.Fuss property*), 126  

```
pretty_exception()
```

 (*in module nitpick.exceptions*), 77  

```
Project
```

 (*class in nitpick.core*), 71  

```
project
```

 (*nitpick.core.Nitpick attribute*), 71  

```
project
```

 (*nitpick.Nitpick attribute*), 37  

```
project
```

 (*nitpick.plugins.info.FileInfo attribute*), 39  

```
project
```

 (*nitpick.style.ConfigValidator attribute*), 114  

```
project
```

 (*nitpick.style.StyleManager attribute*), 124  

```
ProjectViolations
```

 (*class in nitpick.violations*), 126  

```
protocols
```

 (*nitpick.style.FileFetcher attribute*), 115  

```
protocols
```

 (*nitpick.style.GitHubFetcher attribute*), 115  

```
protocols
```

 (*nitpick.style.HttpFetcher attribute*), 117  

```
protocols
```

 (*nitpick.style.PythonPackageFetcher attribute*), 118  

```
protocols
```

 (*nitpick.style.StyleFetcher attribute*), 123  

```
PY
```

 (*nitpick.style.Scheme attribute*), 118  

```
PYPACKAGE
```

 (*nitpick.style.Scheme attribute*), 118  

```
PythonPackageFetcher
```

 (*class in nitpick.style*), 117  

```
PythonPackageURL
```

 (*class in nitpick.style*), 118
- ## Q
- ```
query_params
```

 (*nitpick.style.GitHubURL attribute*), 116

```
QUESTION_MARK
```

 (*nitpick.constants.EmojiEnum attribute*), 70

```
QuitComplainingError
```

, 76

```
quote_if_dotted()
```

 (*in module nitpick.blender*), 65

```
quote_reducer()
```

 (*in module nitpick.blender*), 66

```
quoted_split()
```

 (*in module nitpick.blender*), 66

```
quotes_splitter()
```

 (*in module nitpick.blender*), 66
- ## R
- ```
raw_content_url
```

 (*nitpick.style.GitHubURL property*), 116  

```
read_configuration()
```

 (*nitpick.core.Project method*), 72  

```
reader
```

 (*nitpick.blender.SensibleYAML property*), 64  

```
real
```

 (*nitpick.constants.CachingEnum attribute*), 69  

```
rebuild_dynamic_schema()
```

 (*nitpick.style.StyleManager method*), 125  

```
reformatted
```

 (*nitpick.blender.BaseDoc property*), 61  

```
reformatted
```

 (*nitpick.blender.JsonDoc property*), 62  

```
reformatted
```

 (*nitpick.blender.TomlDoc property*), 65  

```
reformatted
```

 (*nitpick.blender.YamlDoc property*), 65  

```
register_class()
```

 (*nitpick.blender.SensibleYAML method*), 64  

```
relative_to_current_dir()
```

 (*in module nitpick.generic*), 86  

```
removeprefix()
```

 (*nitpick.style.Scheme method*), 121  

```
removesuffix()
```

 (*nitpick.style.Scheme method*), 121  

```
replace
```

 (*nitpick.blender.Comparison property*), 61  

```
replace()
```

 (*nitpick.style.Scheme method*), 121  

```
replace_or_add_list_element()
```

 (*in module nitpick.blender*), 66  

```
repo_root()
```

 (*in module nitpick.style*), 125  

```
report()
```

 (*nitpick.plugins.json.JsonPlugin method*), 47  

```
report()
```

 (*nitpick.plugins.toml.TomlPlugin method*), 57  

```
report()
```

 (*nitpick.plugins.yaml.YamlPlugin method*), 59  

```
Reporter
```

 (*class in nitpick.violations*), 126  

```
repository
```

 (*nitpick.style.GitHubURL attribute*), 116  

```
representer
```

 (*nitpick.blender.SensibleYAML property*), 64  

```
requires_connection
```

 (*nitpick.style.FileFetcher attribute*), 115  

```
requires_connection
```

 (*nitpick.style.GitHubFetcher attribute*), 115  

```
requires_connection
```

 (*nitpick.style.HttpFetcher attribute*), 117  

```
requires_connection
```

 (*nitpick.style.PythonPackageFetcher attribute*), 118  

```
requires_connection
```

 (*nitpick.style.StyleFetcher attribute*), 123  

```
reset()
```

 (*nitpick.violations.Reporter class method*), 127  

```
resolver
```

 (*nitpick.blender.SensibleYAML property*), 64  

```
resource_name
```

 (*nitpick.style.PythonPackageURL attribute*), 118  

```
rfind()
```

 (*nitpick.style.Scheme method*), 121  

```
rindex()
```

 (*nitpick.style.Scheme method*), 121  

```
rjust()
```

 (*nitpick.style.Scheme method*), 122  

```
root
```

 (*nitpick.core.Project property*), 72  

```
root
```

 (*nitpick.fields.Dict attribute*), 78  

```
root
```

 (*nitpick.fields.Field attribute*), 80  

```
root
```

 (*nitpick.fields.List attribute*), 81  

```
root
```

 (*nitpick.fields.Nested attribute*), 83  

```
root
```

 (*nitpick.fields.String attribute*), 85  

```
rpartition()
```

 (*nitpick.style.Scheme method*), 122

rsplit() (*nitpick.style.Scheme* method), 122  
 rstrip() (*nitpick.style.Scheme* method), 122  
 run() (*nitpick.core.Nitpick* method), 71  
 run() (*nitpick.flake8.NitpickFlake8Extension* method), 85  
 run() (*nitpick.Nitpick* method), 38

## S

scalar (*nitpick.blender.ElementDetail* attribute), 62  
 scan() (*nitpick.blender.SensibleYAML* method), 64  
 scanner (*nitpick.blender.SensibleYAML* property), 64  
 schema (*nitpick.fields.Nested* property), 83  
 Scheme (class in *nitpick.style*), 118  
 schemas (*nitpick.style.StyleFetcherManager* attribute), 124  
 search\_json() (in module *nitpick.blender*), 66  
 SensibleYAML (class in *nitpick.blender*), 63  
 SEPARATOR\_QUOTED\_SPLIT (in module *nitpick.blender*), 63  
 seq() (*nitpick.blender.SensibleYAML* method), 64  
 serialize() (*nitpick.blender.SensibleYAML* method), 64  
 serialize() (*nitpick.fields.Dict* method), 78  
 serialize() (*nitpick.fields.Field* method), 80  
 serialize() (*nitpick.fields.List* method), 81  
 serialize() (*nitpick.fields.Nested* method), 83  
 serialize() (*nitpick.fields.String* method), 85  
 serialize\_all() (*nitpick.blender.SensibleYAML* method), 64  
 serializer (*nitpick.blender.SensibleYAML* property), 64  
 session (*nitpick.style.FileFetcher* attribute), 115  
 session (*nitpick.style.GitHubFetcher* attribute), 116  
 session (*nitpick.style.HttpFetcher* attribute), 117  
 session (*nitpick.style.PythonPackageFetcher* attribute), 118  
 session (*nitpick.style.StyleFetcher* attribute), 123  
 session (*nitpick.style.StyleFetcherManager* attribute), 124  
 set\_class (*nitpick.core.ToolNitpickSectionSchema* attribute), 75  
 set\_class (*nitpick.plugins.json.JsonFileSchema* attribute), 46  
 set\_class (*nitpick.plugins.text.TextItemSchema* attribute), 51  
 set\_class (*nitpick.plugins.text.TextSchema* attribute), 56  
 set\_class (*nitpick.schemas.BaseNitpickSchema* attribute), 90  
 set\_class (*nitpick.schemas.BaseStyleSchema* attribute), 94  
 set\_class (*nitpick.schemas.IniSchema* attribute), 98  
 set\_class (*nitpick.schemas.NitpickFilesSectionSchema* attribute), 101

set\_class (*nitpick.schemas.NitpickMetaSchema* attribute), 105  
 set\_class (*nitpick.schemas.NitpickSectionSchema* attribute), 109  
 set\_class (*nitpick.schemas.NitpickStylesSectionSchema* attribute), 113  
 set\_key\_if\_not\_empty() (in module *nitpick.blender*), 67  
 SharedViolations (class in *nitpick.violations*), 127  
 short\_protocol\_url (*nitpick.style.GitHubURL* property), 116  
 show\_missing\_keys() (*nitpick.plugins.ini.IniPlugin* method), 41  
 singleton() (*nitpick.core.Nitpick* class method), 71  
 singleton() (*nitpick.Nitpick* class method), 38  
 skip\_empty\_suggestion (*nitpick.plugins.base.NitpickPlugin* attribute), 39  
 skip\_empty\_suggestion (*nitpick.plugins.ini.IniPlugin* attribute), 41  
 skip\_empty\_suggestion (*nitpick.plugins.json.JsonPlugin* attribute), 47  
 skip\_empty\_suggestion (*nitpick.plugins.text.TextPlugin* attribute), 52  
 skip\_empty\_suggestion (*nitpick.plugins.toml.TomlPlugin* attribute), 57  
 skip\_empty\_suggestion (*nitpick.plugins.yaml.YamlPlugin* attribute), 59  
 SLEEPY\_FACE (*nitpick.constants.EmojiEnum* attribute), 70  
 SpecialConfig (class in *nitpick.config*), 68  
 split() (*nitpick.style.Scheme* method), 122  
 splitlines() (*nitpick.style.Scheme* method), 122  
 STAR\_CAKE (*nitpick.constants.EmojiEnum* attribute), 70  
 startswith() (*nitpick.style.Scheme* method), 122  
 String (class in *nitpick.fields*), 84  
 strip() (*nitpick.style.Scheme* method), 122  
 StyleFetcher (class in *nitpick.style*), 123  
 StyleFetcherManager (class in *nitpick.style*), 123  
 StyleManager (class in *nitpick.style*), 124  
 styles (*nitpick.core.Configuration* attribute), 70  
 StyleViolations (class in *nitpick.violations*), 127  
 suggest\_styles() (*nitpick.core.Project* method), 72  
 suggestion (*nitpick.violations.Fuss* attribute), 126  
 swapcase() (*nitpick.style.Scheme* method), 122

## T

table (*nitpick.core.Configuration* attribute), 70  
 tags (*nitpick.plugins.info.FileInfo* attribute), 39  
 TextItemSchema (class in *nitpick.plugins.text*), 47  
 TextItemSchema.Meta (class in *nitpick.plugins.text*), 47

- TextPlugin (class in *nitpick.plugins.text*), 51
- TextSchema (class in *nitpick.plugins.text*), 52
- TextSchema.Meta (class in *nitpick.plugins.text*), 52
- title() (*nitpick.style.Scheme* method), 123
- to\_bytes() (*nitpick.constants.CachingEnum* method), 69
- token (*nitpick.style.GitHubURL* property), 116
- TomlDoc (class in *nitpick.blender*), 65
- TomlPlugin (class in *nitpick.plugins.toml*), 57
- ToolNitpickSectionSchema (class in *nitpick.core*), 72
- ToolNitpickSectionSchema.Meta (class in *nitpick.core*), 72
- TOP\_SECTION\_HAS\_DIFFERENT\_VALUE (*nitpick.plugins.ini.Violations* attribute), 42
- TOP\_SECTION\_MISSING\_OPTION (*nitpick.plugins.ini.Violations* attribute), 42
- translate() (*nitpick.style.Scheme* method), 123
- traverse\_toml\_tree() (in module *nitpick.blender*), 67
- traverse\_yaml\_tree() (in module *nitpick.blender*), 67
- TYPE\_MAPPING (*nitpick.core.ToolNitpickSectionSchema* attribute), 73
- TYPE\_MAPPING (*nitpick.plugins.json.JsonFileSchema* attribute), 43
- TYPE\_MAPPING (*nitpick.plugins.text.TextItemSchema* attribute), 48
- TYPE\_MAPPING (*nitpick.plugins.text.TextSchema* attribute), 53
- TYPE\_MAPPING (*nitpick.schemas.BaseNitpickSchema* attribute), 87
- TYPE\_MAPPING (*nitpick.schemas.BaseStyleSchema* attribute), 91
- TYPE\_MAPPING (*nitpick.schemas.IniSchema* attribute), 95
- TYPE\_MAPPING (*nitpick.schemas.NitpickFilesSectionSchema* attribute), 99
- TYPE\_MAPPING (*nitpick.schemas.NitpickMetaSchema* attribute), 103
- TYPE\_MAPPING (*nitpick.schemas.NitpickSectionSchema* attribute), 107
- TYPE\_MAPPING (*nitpick.schemas.NitpickStylesSectionSchema* attribute), 111
- U**
- update\_comment\_before() (in module *nitpick.tomlkit\_ext*), 125
- updater (*nitpick.blender.YamlDoc* attribute), 65
- updater (*nitpick.plugins.ini.IniPlugin* attribute), 41
- upper() (*nitpick.style.Scheme* method), 123
- URL (in module *nitpick.fields*), 85
- url (*nitpick.style.BuiltinStyle* attribute), 114
- url (*nitpick.style.GitHubURL* property), 116
- url\_to\_python\_path() (in module *nitpick.generic*), 86
- V**
- validate() (*nitpick.core.ToolNitpickSectionSchema* method), 75
- validate() (*nitpick.plugins.json.JsonFileSchema* method), 46
- validate() (*nitpick.plugins.text.TextItemSchema* method), 51
- validate() (*nitpick.plugins.text.TextSchema* method), 56
- validate() (*nitpick.schemas.BaseNitpickSchema* method), 90
- validate() (*nitpick.schemas.BaseStyleSchema* method), 94
- validate() (*nitpick.schemas.IniSchema* method), 98
- validate() (*nitpick.schemas.NitpickFilesSectionSchema* method), 102
- validate() (*nitpick.schemas.NitpickMetaSchema* method), 105
- validate() (*nitpick.schemas.NitpickSectionSchema* method), 109
- validate() (*nitpick.schemas.NitpickStylesSectionSchema* method), 113
- validate() (*nitpick.style.ConfigValidator* method), 114
- validation\_schema (*nitpick.plugins.base.NitpickPlugin* attribute), 39
- validation\_schema (*nitpick.plugins.ini.IniPlugin* attribute), 41
- validation\_schema (*nitpick.plugins.json.JsonPlugin* attribute), 47
- validation\_schema (*nitpick.plugins.text.TextPlugin* attribute), 52
- validation\_schema (*nitpick.plugins.toml.TomlPlugin* attribute), 57
- validation\_schema (*nitpick.plugins.yaml.YamlPlugin* attribute), 59
- value (*nitpick.config.OverridableConfig* attribute), 68
- version (*nitpick.blender.SensibleYAML* property), 65
- version (*nitpick.flake8.NitpickFlake8Extension* attribute), 85
- violation\_base\_code (*nitpick.plugins.base.NitpickPlugin* attribute), 39
- violation\_base\_code (*nitpick.plugins.ini.IniPlugin* attribute), 41
- violation\_base\_code (*nitpick.plugins.json.JsonPlugin* attribute), 47
- violation\_base\_code (*nitpick.plugins.text.TextPlugin* attribute), 52
- violation\_base\_code (*nitpick.plugins.toml.TomlPlugin* attribute), 57
- violation\_base\_code (*nitpick.plugins.yaml.YamlPlugin* attribute), 59
- ViolationEnum (class in *nitpick.violations*), 127

Violations (class in *nitpick.plugins.ini*), 41  
Violations (class in *nitpick.plugins.text*), 56

## W

`with_traceback()` (*nitpick.exceptions.QuitComplainingError* method), 77  
`write_file()` (*nitpick.plugins.base.NitpickPlugin* method), 39  
`write_file()` (*nitpick.plugins.ini.IniPlugin* method), 41  
`write_file()` (*nitpick.plugins.json.JsonPlugin* method), 47  
`write_file()` (*nitpick.plugins.text.TextPlugin* method), 52  
`write_file()` (*nitpick.plugins.toml.TomlPlugin* method), 58  
`write_file()` (*nitpick.plugins.yaml.YamlPlugin* method), 59  
`write_initial_contents()` (*nitpick.plugins.base.NitpickPlugin* method), 39  
`write_initial_contents()` (*nitpick.plugins.ini.IniPlugin* method), 41  
`write_initial_contents()` (*nitpick.plugins.json.JsonPlugin* method), 47  
`write_initial_contents()` (*nitpick.plugins.text.TextPlugin* method), 52  
`write_initial_contents()` (*nitpick.plugins.toml.TomlPlugin* method), 58  
`write_initial_contents()` (*nitpick.plugins.yaml.YamlPlugin* method), 59

## X

`X_RED_CROSS` (*nitpick.constants.EmojiEnum* attribute), 70  
`Xdump_all()` (*nitpick.blender.SensibleYAML* method), 63

## Y

`YamlDoc` (class in *nitpick.blender*), 65  
`YamlPlugin` (class in *nitpick.plugins.yaml*), 58

## Z

`zfill()` (*nitpick.style.Scheme* method), 123