nitpick Documentation

Release 0.22.2

W. Augusto Andreoli

Jul 05, 2020

CONTENTS:

Flake8 plugin to enforce the same tool configuration (flake8, isort, mypy, Pylint...) across multiple Python projects.

Useful if you maintain multiple projects and want to use the same configs in all of them.

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 style changes*.

CHAPTER

ONE

INSTALLATION GUIDE

1.1 Quick setup

To try the package, simply install it (in a virtualenv or globally) and run flake8 on a project with at least one Python (`.py`) file:

```
$ cd /path/to/my/python/project

# Install using pip:
$ pip install -U nitpick

# Or using Poetry:
$ poetry add --dev nitpick

$ flake8 .
```

Nitpick will download and use the opinionated default style file.

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

1.2 Run as a pre-commit hook (recommended)

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

```
repos:
  - repo: https://github.com/andreoliwa/nitpick
    rev: v0.22.2
    hooks:
      - id: 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
```

CHAPTER

TWO

STYLES

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 *the [tool.nitpick] section* for details.

You can also check the *pre-configured defaults*, and copy/paste/change configuration from them.

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 *The [tool.nitpick] section*.
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 Breaking style 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.

2.5.1 `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"
```

CHAPTER

THREE

AUTO-DETECTION

3.1 Root dir of the project

Nitpick tries to find the root dir of the project using some hardcoded assumptions.

1. Starting from the current working directory, it will search for files that are usually in the root of a Python project:
   - `pyproject.toml`
   - `setup.py`
   - `setup.cfg`
   - `requirements*.txt`
   - `Pipfile` (Pipenv)
   - `app.py` and `wsgi.py` (Flask CLI)
   - `autoapp.py`
2. If none of these root files were found, search for `manage.py`. On Django projects, it can be in another dir inside the root dir (issue 21).
3. If multiple roots are found, get the top one in the dir tree.

3.2 Main Python file

After finding 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

CHAPTER

FOUR

THE [TOOL.NITPICK] SECTION

The style file for your project should be configured in the `[tool.nitpick]` section of your `pyproject.toml` file.

You can configure your own style like this:

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

You can set `style` with any local file or URL. E.g.: you can use the raw URL of a GitHub Gist.

Using a file in your home directory:

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

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

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

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.

CHAPTER

FIVE

THE [NITPICK] SECTION

The [nitpick] section 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:

```
[nitpick.styles]
include = ["styles/python37", "styles/poetry"]
```

The styles will be merged following the sequence in the list.

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

5.4 [nitpick.JSONFile]

Configure the list of filenames that should be checked by the *`nitpick.plugins.json.JSONFile`* class. See *the default package.json style* for an example of usage.

CHAPTER

SIX

CONFIGURATION FILES

Below are the currently supported configuration files.

6.1 pyproject.toml

Checker for pyproject.toml.

See also PEP 518.

Example: *the Python 3.7 default*. There are many other examples in *Defaults*.

6.2 setup.cfg

Checker for the setup.cfg config file.

Example: *flake8 configuration*.

6.3 .pre-commit-config.yaml

Checker for the .pre-commit-config.yaml file.

Example: *the default pre-commit hooks*.

6.4 JSON files

Checker for any JSON file.

First, configure the list of files to be checked in the *[nitpick.JSONFile] section*.

Then add the configuration for the file name you just declared. Example: *the default config for package.json*.

If a JSON file is configured on `[nitpick.JSONFile] file_names`, then a configuration for it should exist. Otherwise, a style validation error will be raised.

CHAPTER

SEVEN

DEFAULTS

If you don't *configure your own style*, those are some of the defaults that will be applied.

All TOML configs below are taken from the default style file.

7.1 Absent files

Content of styles/absent-files.toml:

```
[nitpick.files.absent]
"requirements.txt" = "Install poetry, run 'poetry init' to create pyproject.toml, and␣
↪move dependencies to it"
".isort.cfg" = "Move values to setup.cfg, section [isort]"
"Pipfile" = "Use pyproject.toml instead"
"Pipfile.lock" = "Use pyproject.toml instead"
".venv" = ""
".pyup.yml" = "Configure .travis.yml with safety instead: https://github.com/pyupio/
↪safety#using-safety-with-a-ci-service"
```

7.2 black

Content of styles/black.toml:

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

[["pre-commit-config.yaml".repos]]
yaml = """
  - repo: https://github.com/python/black
    rev: 19.10b0
    hooks:
      - id: black
        args: [--safe, --quiet]
  - repo: https://github.com/asottile/blacken-docs
    rev: v1.7.0
    hooks:
      - id: blacken-docs
        additional_dependencies: [black==19.10b0]
"""
# TODO The toml library has issues loading arrays with multiline strings:
```

(continues on next page)

(continued from previous page)

```
#  https://github.com/uiri/toml/issues/123
#  https://github.com/uiri/toml/issues/230
#  If they are fixed one day, remove this 'yaml' key and use only a 'repos' list with␣
↪a single element:
#["pre-commit-config.yaml"]
#repos = ["""
#<YAML goes here>
#"""]
```

7.3 flake8

Content of styles/flake8.toml:

```
["setup.cfg".flake8]
# http://www.pydocstyle.org/en/2.1.1/error_codes.html
# Ignoring W503: https://github.com/python/black#line-breaks--binary-operators
# Ignoring "D202 No blank lines allowed after function docstring": black inserts a␣
↪blank line.
ignore = "D107,D202,D203,D401,E203,E402,E501,W503"
max-line-length = 120
inline-quotes = "double"
exclude = ".tox,build"

# Nitpick recommends those plugins as part of the style, but doesn't install them␣
↪automatically as before.
# This way, the developer has the choice of overriding this style, instead of having␣
↪lots of plugins installed
# without being asked.
[["pre-commit-config.yaml".repos]]
yaml = """
  - repo: https://gitlab.com/pycqa/flake8
    rev: 3.8.3
    hooks:
      - id: flake8
        additional_dependencies: [flake8-blind-except, flake8-bugbear, flake8-
↪comprehensions,
          flake8-debugger, flake8-docstrings, flake8-isort, flake8-polyfill,
          flake8-pytest, flake8-quotes, yesqa]
"""
# TODO suggest nitpick for external repos
```

7.4 IPython

Content of styles/ipython.toml:

```
["pyproject.toml".tool.poetry.dev-dependencies]
ipython = "*"
ipdb = "*"
```

7.5 isort

Content of styles/isort.toml:

```
["setup.cfg".isort]
line_length = 120
skip = ".tox,build"
known_first_party = "tests"

# The configuration below is needed for compatibility with black.
# https://github.com/python/black#how-black-wraps-lines
# https://github.com/timothycrosley/isort#multi-line-output-modes
multi_line_output = 3
include_trailing_comma = true
force_grid_wrap = 0
combine_as_imports = true

[["pre-commit-config.yaml".repos]]
yaml = """
  - repo: https://github.com/asottile/seed-isort-config
    rev: v2.2.0
    hooks:
      - id: seed-isort-config
  - repo: https://github.com/pre-commit/mirrors-isort
    rev: v4.3.21
    hooks:
      - id: isort
"""
```

7.6 mypy

Content of styles/mypy.toml:

```
# https://mypy.readthedocs.io/en/latest/config_file.html
["setup.cfg".mypy]
ignore_missing_imports = true

# Do not follow imports (except for ones found in typeshed)
follow_imports = "skip"

# Treat Optional per PEP 484
strict_optional = true

# Ensure all execution paths are returning
warn_no_return = true

# Lint-style cleanliness for typing
warn_redundant_casts = true
warn_unused_ignores = true

[["pre-commit-config.yaml".repos]]
yaml = """
  - repo: https://github.com/pre-commit/mirrors-mypy
    rev: v0.782
    hooks:
```

(continues on next page)

(continued from previous page)

```
      - id: mypy
"""
```

7.7 package.json

Content of styles/package-json.toml:

```
[nitpick.JSONFile]
file_names = ["package.json"]

["package.json"]
contains_keys = ["name", "version", "repository.type", "repository.url", "release.
↪plugins"]

["package.json".contains_json]
commitlint = """
  {
    "extends": [
      "@commitlint/config-conventional"
    ]
  }
"""
```

7.8 Poetry

Content of styles/poetry.toml:

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

7.9 Bash

Content of styles/pre-commit/bash.toml:

```
[["pre-commit-config.yaml".repos]]
yaml = """
  - repo: https://github.com/openstack/bashate
    rev: 2.0.0
    hooks:
      - id: bashate
"""
```

7.10 commitlint

Content of styles/pre-commit/commitlint.toml:

```
[["pre-commit-config.yaml".repos]]
yaml = """
  - repo: https://github.com/alessandrojcm/commitlint-pre-commit-hook
    rev: v2.2.3
    hooks:
      - id: commitlint
        stages: [commit-msg]
        additional_dependencies: ['@commitlint/config-conventional']
"""
```

7.11 pre-commit (hooks)

Content of styles/pre-commit/general.toml:

```
[["pre-commit-config.yaml".repos]]
yaml = """
  - repo: https://github.com/pre-commit/pre-commit-hooks
    rev: v3.1.0
    hooks:
      - id: debug-statements
      - id: end-of-file-fixer
      - id: trailing-whitespace
  - repo: https://github.com/asottile/pyupgrade
    rev: v2.6.2
    hooks:
      - id: pyupgrade
"""
```

7.12 pre-commit (main)

Content of styles/pre-commit/main.toml:

```
# See https://pre-commit.com for more information
# See https://pre-commit.com/hooks.html for more hooks

[nitpick.files.present]
".pre-commit-config.yaml" = "Create the file with the contents below, then run 'pre-
↪commit install'"
```

7.13 pre-commit (Python hooks)

Content of styles/pre-commit/python.toml:

```
[["pre-commit-config.yaml".repos]]
yaml = """
  - repo: https://github.com/pre-commit/pygrep-hooks
    rev: v1.5.1
    hooks:
      - id: python-check-blanket-noqa
      - id: python-check-mock-methods
      - id: python-no-eval
      - id: python-no-log-warn
      - id: rst-backticks
"""
```

7.14 Pylint

Content of styles/pylint.toml:

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

7.15 Python 3.5, 3.6, 3.7 to 3.8

Content of styles/python35-36-37-38.toml:

```
["pyproject.toml".tool.poetry.dependencies]
python = "^3.5 || ^3.6 || ^3.7 || ^3.8"
```

7.16 Python 3.5, 3.6 or 3.7

Content of styles/python35-36-37.toml:

```
["pyproject.toml".tool.poetry.dependencies]
python = "^3.5 || ^3.6 || ^3.7"
```

7.17 Python 3.6 or 3.7

Content of styles/python36-37.toml:

```
["pyproject.toml".tool.poetry.dependencies]
python = "^3.6 || ^3.7"
```

7.18 Python 3.6

Content of styles/python36.toml:

```
["pyproject.toml".tool.poetry.dependencies]
python = "^3.6"
```

7.19 Python 3.7

Content of styles/python37.toml:

```
["pyproject.toml".tool.poetry.dependencies]
python = "^3.7"
```

CHAPTER

EIGHT

TROUBLESHOOTING

8.1 Crash on multi-threading

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

```
objc[93329]: +[__NSPlaceholderDate initialize] may have been in progress in another␣
↪thread when fork() was called.
objc[93329]: +[__NSPlaceholderDate 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.

CHAPTER

NINE

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.

9.1 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 there.

9.2 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.

9.3 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. Create your virtualenv with pyenv (or some other tool you prefer):

```
pyenv virtualenv 3.5.6 nitpick
pyenv activate nitpick
```

5. Install packages:

```
poetry install

# Output:
# Installing dependencies from lock file
# ...
```

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 pre-commit checks and tests with:

```
make
```

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

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

9. Submit a pull request through the GitHub website.
10. If your pull request is accepted, all your commits will be squashed into one, and the Conventional Commits Format will be used on the commit message.

9.3.1 Pull Request Guidelines

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

For merging, you should:

1. Include passing tests (run `make test`)[1].
2. Update documentation when there's new API, functionality etc.
3. Add yourself to `AUTHORS.rst`.

[1] If you don't have all the necessary python versions available locally you can rely on Travis - it will run the tests for each change you add in the pull request.
It will be slower though ...

CHAPTER

TEN

AUTHORS

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

CHAPTER

ELEVEN

NITPICK

11.1 nitpick package

Main module.

11.1.1 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.

nitpick.plugins.**handle_config_file**(*config: Dict[str, Any]*, *file_name: str*, *tags: Set[str]*) → Optional[BaseFile]

Return a BaseFile if this plugin handles the relative filename or any of its **:py:package:`identify`** tags.

Submodules

nitpick.plugins.base module

Base class for file checkers.

class nitpick.plugins.base.**BaseFile**(*config: Dict[str, Any]*, *file_name: str = None*)

Bases: *nitpick.mixin.NitpickMixin*

Base class for file checkers.

check_exists() → Iterator[Tuple[int, int, str, Type]]

Check if the file should exist.

abstract check_rules() → Iterator[Tuple[int, int, str, Type]]

Check rules for this file. It should be overridden by inherited classes if needed.

dynamic_name_classes: Set[Type[BaseFile]] = {}

error_base_number = 300

error_prefix = ''

file_name = ''

fixed_name_classes: Set[Type[BaseFile]] = {}

flake8_error(*number: int*, *message: str*, *suggestion: str = None*, *add_to_base_number=True*) → Tuple[int, int, str, Type]
Return a flake8 error as a tuple.

classmethod get_compiled_jmespath_file_names()
Return a compiled JMESPath expression for file names, using the class name as part of the key.

identify_tags: Set[str] = {}
Which **:py:package:`identify`** tags this `nitpick.files.base.BaseFile` child recognises.

classmethod load_fixed_dynamic_classes() → None
Separate classes with fixed file names from classes with dynamic files names.

nested_field: Optional[Schema] = 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.JSONFile`*).

abstract suggest_initial_contents() → str
Suggest the initial content for this missing file.

warn_missing_different(*comparison:* nitpick.formats.Comparison, *prefix_message: str = ''*)
Warn about missing and different keys.

nitpick.plugins.json module

JSON files.

class `nitpick.plugins.json.`**JSONFile**(*config: Dict[str, Any]*, *file_name: str = None*)
Bases: *`nitpick.plugins.base.BaseFile`*

Checker for any JSON file.

First, configure the list of files to be checked in the *[nitpick.JSONFile] section*.

Then add the configuration for the file name you just declared. Example: *the default config for package.json*.

If a JSON file is configured on `[nitpick.JSONFile] file_names`, then a configuration for it should exist. Otherwise, a style validation error will be raised.

SOME_VALUE_PLACEHOLDER = '<some value here>'

check_exists() → Iterator[Tuple[int, int, str, Type]]
Check if the file should exist.

check_rules() → Iterator[Tuple[int, int, str, Type]]
Check missing keys and JSON content.

dynamic_name_classes = {}

error_base_number = 340

error_prefix = ''

file_name = ''

fixed_name_classes = {}

flake8_error(*number: int*, *message: str*, *suggestion: str = None*, *add_to_base_number=True*) → Tuple[int, int, str, Type]
Return a flake8 error as a tuple.

classmethod get_compiled_jmespath_file_names()
Return a compiled JMESPath expression for file names, using the class name as part of the key.

get_suggested_json(*raw_actual: Dict[str, Any] = None*) → Dict[str, Any]
Return the suggested JSON based on actual values.

identify_tags = {'json'}

classmethod load_fixed_dynamic_classes() → None
Separate classes with fixed file names from classes with dynamic files names.

nested_field
alias of *JSONFileSchema*

suggest_initial_contents() → str
Suggest the initial content for this missing file.

warn_missing_different(*comparison:* nitpick.formats.Comparison, *prefix_message: str = ''*)
Warn about missing and different keys.

class nitpick.plugins.json.**JSONFileSchema**(*, *only: Union[Sequence[str], Set[str]] = None*, *exclude: Union[Sequence[str], Set[str]] = ()*, *many: bool = False*, *context: Dict = None*, *load_only: Union[Sequence[str], Set[str]] = ()*, *dump_only: Union[Sequence[str], Set[str]] = ()*, *partial: Union[bool, Sequence[str], Set[str]] = False*, *unknown: str = None*)
Bases: *nitpick.schemas.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.
- **`render_module`: Module to use for *loads <Schema.loads>* and *dumps <Schema.dumps>*.** Defaults to *json* from the standard library.

- **ordered: If *True*, order serialization output according to the** order in which fields were declared. 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 `marshmallow.schema.SchemaOpts`

TYPE_MAPPING = {<class 'str'>: <class 'marshmallow.fields.String'>, <class 'bytes'>:

property dict_class

dump(*obj: Any*, *, *many: bool = 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 A dict of serialized data

Return type dict

New in version 1.0.0.

Changed in version 3.0.0b7: This method returns the serialized data rather than a `(data, errors)` duple. 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*, ***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

Return type str

New in version 1.0.0.

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

error_messages = {'unknown': 'Unknown configuration. See https://nitpick.rtfd.io/en/l

classmethod from_dict(*fields: Dict[str, Union[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: marshmallow.exceptions.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: Union[Mapping[str, Any], Iterable[Mapping[str, Any]]]*, ***, *many: bool = None*, *partial: Union[bool, Sequence[str], Set[str]] = None*, *unknown: str = 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)` duple. A `ValidationError` is raised if invalid data are passed.

loads(*json_data: str*, ***, *many: bool = None*, *partial: Union[bool, Sequence[str], Set[str]] = None*, *unknown: str = 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)` duple. A `ValidationError` is raised if invalid data are passed.

on_bind_field(*field_name: str*, *field_obj: marshmallow.fields.Field*) → None
Hook to modify a field when it is bound to the *Schema*.

No-op by default.

opts = <marshmallow.schema.SchemaOpts object>

property set_class

validate(*data: Mapping*, ***, *many: bool = None*, *partial: Union[bool, Sequence[str], Set[str]] = 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.plugins.json.**handle_config_file**(*config: Dict[str, Any]*, *file_name: str*, *tags: Set[str]*) → Optional[*nitpick.plugins.base.BaseFile*]
Handle JSON files.

nitpick.plugins.pre_commit module

Checker for the .pre-commit-config.yaml file.

class nitpick.plugins.pre_commit.**PreCommitFile**(*config: Dict[str, Any]*, *file_name: str = None*)

Bases: *nitpick.plugins.base.BaseFile*

Checker for the .pre-commit-config.yaml file.

Example: *the default pre-commit hooks*.

actual_hooks: OrderedDict[str, PreCommitHook] = {}

actual_hooks_by_index: List[str] = []

actual_hooks_by_key: Dict[str, int] = {}

actual_yaml: YamlFormat = None

check_exists() → Iterator[Tuple[int, int, str, Type]]
Check if the file should exist.

check_hooks() → Iterator[Tuple[int, int, str, Type]]
Check the repositories configured in pre-commit.

check_repo_block(*expected_repo_block: collections.OrderedDict*) → Iterator[Tuple[int, int, str, Type]]
Check a repo with a YAML string configuration.

check_repo_old_format(*index: int*, *repo_data: collections.OrderedDict*) → Iterator[Tuple[int, int, str, Type]]
Check repos using the old deprecated format with `hooks` and `repo` keys.

check_rules() → Iterator[Tuple[int, int, str, Type]]
Check the rules for the pre-commit hooks.

dynamic_name_classes = {}

error_base_number = 330

error_prefix = ''

file_name = '.pre-commit-config.yaml'

fixed_name_classes = {}

flake8_error(*number: int*, *message: str*, *suggestion: str = None*, *add_to_base_number=True*) → Tuple[int, int, str, Type]
Return a flake8 error as a tuple.

static format_hook(*expected_dict*) → str
Format the hook so it's easy to copy and paste it to the .yaml file: ID goes first, indent with spaces.

classmethod get_compiled_jmespath_file_names()
Return a compiled JMESPath expression for file names, using the class name as part of the key.

identify_tags = {}

classmethod load_fixed_dynamic_classes() → None
Separate classes with fixed file names from classes with dynamic files names.

nested_field = None

suggest_initial_contents() → str
Suggest the initial content for this missing file.

warn_missing_different(*comparison:* nitpick.formats.Comparison, *prefix_message: str* = '')
Warn about missing and different keys.

class nitpick.plugins.pre_commit.**PreCommitHook**(*repo: str*, *hook_id: str*, *yaml:* nitpick.formats.YamlFormat)
Bases: object

A pre-commit hook.

classmethod get_all_hooks_from(*str_or_yaml: Union[str, ruamel.yaml.comments.CommentedSeq, ruamel.yaml.comments.CommentedMap]*)
Get all hooks from a YAML string. Split the string in hooks and copy the repo info for each.

property key_value_pair
Key/value pair to be used as a dict item.

property single_hook
Return only a single hook instead of a list.

property unique_key
Unique key of this hook, to be used in a dict.

nitpick.plugins.pre_commit.**handle_config_file**(*config: Dict[str, Any]*, *file_name: str*, *tags: Set[str]*) → Optional[*nitpick.plugins.base.BaseFile*]
Handle pre-commit config file.

nitpick.plugins.pyproject_toml module

Checker for pyproject.toml.

class nitpick.plugins.pyproject_toml.**PyProjectTomlFile**(*config: Dict[str, Any]*, *file_name: str = None*)
Bases: *nitpick.plugins.base.BaseFile*

Checker for pyproject.toml.

See also PEP 518.

Example: *the Python 3.7 default*. There are many other examples in *Defaults*.

check_exists() → Iterator[Tuple[int, int, str, Type]]
Check if the file should exist.

check_rules() → Iterator[Tuple[int, int, str, Type]]
Check missing key/value pairs in pyproject.toml.

dynamic_name_classes = {}

error_base_number = 310

error_prefix = ''

file_name = 'pyproject.toml'

fixed_name_classes = {}

flake8_error(*number: int*, *message: str*, *suggestion: str = None*, *add_to_base_number=True*) → Tuple[int, int, str, Type]
Return a flake8 error as a tuple.

classmethod get_compiled_jmespath_file_names()
Return a compiled JMESPath expression for file names, using the class name as part of the key.

identify_tags = {}

classmethod load_fixed_dynamic_classes() → None
Separate classes with fixed file names from classes with dynamic files names.

nested_field = None

suggest_initial_contents() → str
Suggest the initial content for this missing file.

warn_missing_different(*comparison:* nitpick.formats.Comparison, *prefix_message: str = ''*)
Warn about missing and different keys.

nitpick.plugins.pyproject_toml.**handle_config_file**(*config: Dict[str, Any]*, *file_name: str*, *tags: Set[str]*) → Optional[*nitpick.plugins.base.BaseFile*]
Handle pyproject.toml file.

nitpick.plugins.setup_cfg module

Checker for the *setup.cfg <https://docs.python.org/3/distutils/configfile.html>* config file.

class nitpick.plugins.setup_cfg.**SetupCfgFile**(*config: Dict[str, Any]*, *file_name: str = None*)
Bases: *nitpick.plugins.base.BaseFile*

Checker for the setup.cfg config file.

Example: *flake8 configuration*.

COMMA_SEPARATED_VALUES = 'comma_separated_values'

check_exists() → Iterator[Tuple[int, int, str, Type]]
Check if the file should exist.

check_rules() → Iterator[Tuple[int, int, str, Type]]
Check missing sections and missing key/value pairs in setup.cfg.

compare_different_keys(*section*, *key*, *raw_actual: Any*, *raw_expected: Any*) → Iterator[Tuple[int, int, str, Type]]
Compare different keys, with special treatment when they are lists or numeric.

dynamic_name_classes = {}

error_base_number = 320

error_prefix = ''

expected_sections: Set[str] = {}

file_name = 'setup.cfg'

fixed_name_classes = {}

flake8_error(*number: int*, *message: str*, *suggestion: str = None*, *add_to_base_number=True*) → Tuple[int, int, str, Type]
Return a flake8 error as a tuple.

classmethod get_compiled_jmespath_file_names()
Return a compiled JMESPath expression for file names, using the class name as part of the key.

static get_example_cfg(*config_parser: configparser.ConfigParser*) → str
Print an example of a config parser in a string instead of a file.

get_missing_output(*actual_sections: Set[str] = None*) → str
Get a missing output string example from the missing sections in setup.cfg.

identify_tags = {}

classmethod load_fixed_dynamic_classes() → None
Separate classes with fixed file names from classes with dynamic files names.

missing_sections: Set[str] = {}

nested_field = None

show_missing_keys(*section*, *key*, *values: List[Tuple[str, Any]]*) → Iterator[Tuple[int, int, str, Type]]
Show the keys that are not present in a section.

suggest_initial_contents() → str
Suggest the initial content for this missing file.

warn_missing_different(*comparison:* nitpick.formats.Comparison, *prefix_message: str = ''*)
Warn about missing and different keys.

nitpick.plugins.setup_cfg.**handle_config_file**(*config: Dict[str, Any]*, *file_name: str*, *tags: Set[str]*) → Optional[*nitpick.plugins.base.BaseFile*]
Handle the setup.cfg file.

11.1.2 Submodules

nitpick.app module

The Nitpick application.

class nitpick.app.**NitpickApp**
Bases: object

The Nitpick application.

class Flags(*value*)
Bases: enum.Enum

Flags to be used with flake8 CLI.

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

add_style_error(*file_name: str*, *message: str*, *invalid_data: str = None*) → None
Add a style error to the internal list.

static as_flake8_warning(*nitpick_error:* nitpick.exceptions.NitpickError) → Tuple[int, int, str, Type]
Return a flake8 error as a tuple.

cache_dir: Path = None

clear_cache_dir() → pathlib.Path
Clear the cache directory (on the project root or on the current directory).

config: Config = None

classmethod create_app(*offline=False*) → *nitpick.app.NitpickApp*
Create a single application.

classmethod current()
Get the current app from the stack.

find_main_python_file() → pathlib.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.

static find_root_dir() → pathlib.Path
Find the root dir of the Python project (the one that has one of the `ROOT_FILES`).

Start from the current working dir.

classmethod format_env(*flag: enum.Enum*) → str
Format the name of an environment variable.

classmethod format_flag(*flag: enum.Enum*) → str
Format the name of a flag to be used on the CLI.

classmethod get_env(*flag: enum.Enum*) → str
Get the value of an environment variable.

static load_plugins() → pluggy.manager.PluginManager
Load all defined plugins.

main_python_file: Path = None

plugin_manager: PluginManager = None

root_dir: Path = None

nitpick.config module

Configuration of the plugin.

class nitpick.config.**Config**
Bases: *nitpick.mixin.NitpickMixin*

Plugin configuration, read from the project config.

error_base_number = 200

error_prefix = ''

flake8_error(*number: int*, *message: str*, *suggestion: str = None*, *add_to_base_number=True*) → Tuple[int, int, str, Type]
Return a flake8 error as a tuple.

merge_styles() → Iterator[Tuple[int, int, str, Type]]
Merge one or multiple style files.

validate_pyproject_tool_nitpick() → bool
Validate the `pyroject.toml`'s `[tool.nitpick]` section against a Marshmallow schema.

warn_missing_different(*comparison:* nitpick.formats.Comparison, *prefix_message: str = ''*)
Warn about missing and different keys.

nitpick.constants module

Constants.

nitpick.constants.**SEPARATOR_QUOTED_SPLIT = '#$@'**
Special unique separator for *nitpick.generic.quoted_split()*.

nitpick.exceptions module

Nitpick exceptions.

exception nitpick.exceptions.**NitpickError**(**args: object*)
Bases: Exception

A Nitpick error raise flake8 errors.

add_to_base_number = True

args

error_base_number: int = 0

error_prefix: str = ''

message: str = ''

number: int = 0

suggestion: str = None

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

exception nitpick.exceptions.**NoPythonFile**(*root_dir: pathlib.Path*, **args: object*)
Bases: *nitpick.exceptions.PluginError*

No Python file was found.

add_to_base_number = True

args

error_base_number = 100

error_prefix = ''

message = 'No Python file was found on the root dir and subdir of {!r}'

number = 2

suggestion = None

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

exception nitpick.exceptions.**NoRootDir**(**args: object*)
Bases: *nitpick.exceptions.PluginError*

No root dir found.

add_to_base_number = True

args

error_base_number = 100

error_prefix = ''

message = 'No root dir found (is this a Python project?)'

number = 1

suggestion = None

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

exception nitpick.exceptions.**PluginError**(**args: object*)
Bases: *nitpick.exceptions.NitpickError*

Plugin error.

add_to_base_number = True

args

error_base_number = 100

error_prefix = ''

message = ''

number = 0

suggestion = None

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

exception nitpick.exceptions.**StyleError**(*style_file_name: str*, **args: object*)
Bases: *nitpick.exceptions.NitpickError*

An error in a style file.

add_to_base_number = False

args

error_base_number = 0

error_prefix = ''

message = ''

number = 1

suggestion = None

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

nitpick.fields module

Custom Marshmallow fields and validators.

class nitpick.fields.**Dict**(*keys: Union[marshmallow.fields.Field, type] = None*, *values: Union[marshmallow.fields.Field, type] = None*, ***kwargs*)
Bases: marshmallow.fields.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`.

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

deserialize(*value: Any*, *attr: str = None*, *data: Mapping[str, Any] = 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*) → marshmallow.exceptions.ValidationError
Helper method to make a *ValidationError* with an error message from `self.error_messages`.

mapping_type
alias of `builtins.dict`

name = None

parent = None

property root
Reference to the *Schema* that this field belongs to even if it is buried in a container field (e.g. *List*). Return *None* for unbound fields.

serialize(*attr: str*, *obj: Any*, *accessor: Callable[[Any, str, Any], Any] = 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**(**, default: Any = <marshmallow.missing>, missing: Any = <marshmallow.missing>, data_key: str = None, attribute: str = None, validate: Union[Callable[[Any], Any], Iterable[Callable[[Any], Any]]] = None, required: bool = False, allow_none: bool = None, load_only: bool = False, dump_only: bool = False, error_messages: Dict[str, str] = None, **metadata*)

Bases: marshmallow.base.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

- **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.
- **missing** – 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 `missing=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 arguments to be stored as 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`.

default_error_messages = {'null': 'Field may not be null.', 'required': 'Missing data
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*, *data: Mapping[str, Any] = 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*) → marshmallow.exceptions.ValidationError
Helper method to make a *ValidationError* with an error message from `self.error_messages`.

name = None

parent = None

property root
Reference to the *Schema* that this field belongs to even if it is buried in a container field (e.g. *List*). Return *None* for unbound fields.

serialize(*attr: str*, *obj: Any*, *accessor: Callable[[Any, str, Any], Any] = 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: Union[marshmallow.fields.Field, type]*, ***kwargs*)
Bases: marshmallow.fields.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`.

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

deserialize(*value: Any*, *attr: str = None*, *data: Mapping[str, Any] = 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*) → marshmallow.exceptions.ValidationError
Helper method to make a *ValidationError* with an error message from `self.error_messages`.

name = None

parent = None

property root
Reference to the *Schema* that this field belongs to even if it is buried in a container field (e.g. *List*). Return *None* for unbound fields.

serialize(*attr: str*, *obj: Any*, *accessor: Callable[[Any, str, Any], Any] = 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: Union[marshmallow.base.SchemaABC, type, str, Callable[[], marshmallow.base.SchemaABC]]*, ***, *default: Any = <marshmallow.missing>*, *only: Union[Sequence[str], Set[str]] = None*, *exclude: Union[Sequence[str], Set[str]] = ()*, *many: bool = False*, *unknown: str = None*, ***kwargs*)
Bases: `marshmallow.fields.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** – The Schema class or class name (string) to nest, or `"self"` to nest the `Schema` within itself.
- **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`.

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

deserialize(*value: Any*, *attr: str = None*, *data: Mapping[str, Any] = 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*) → marshmallow.exceptions.ValidationError
Helper method to make a *ValidationError* with an error message from `self.error_messages`.

name = None

parent = None

property root
Reference to the *Schema* that this field belongs to even if it is buried in a container field (e.g. *List*). Return *None* for unbound fields.

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*, ***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**(**, default: Any = <marshmallow.missing>, missing: Any = <marshmallow.missing>, data_key: str = None, attribute: str = None, validate: Union[Callable[[Any], Any], Iterable[Callable[[Any], Any]]] = None, required: bool = False, allow_none: bool = None, load_only: bool = False, dump_only: bool = False, error_messages: Dict[str, str] = None, **metadata*)

Bases: marshmallow.fields.Field

A string field.

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

property context
The context dictionary for the parent `Schema`.

default_error_messages = {'invalid': 'Not a valid string.', 'invalid_utf8': 'Not a v

deserialize(*value: Any*, *attr: str = None*, *data: Mapping[str, Any] = 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*) → marshmallow.exceptions.ValidationError
Helper method to make a *ValidationError* with an error message from `self.error_messages`.

name = None

parent = None

property root
Reference to the *Schema* that this field belongs to even if it is buried in a container field (e.g. *List*). Return *None* for unbound fields.

serialize(*attr: str*, *obj: Any*, *accessor: Callable[[Any, str, Any], Any] = 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.flake8 module

Flake8 plugin to check files.

class nitpick.flake8.**NitpickExtension**(*tree=None*, *filename='(none)'*)
Bases: *nitpick.mixin.NitpickMixin*

Main class for the flake8 extension.

static add_options(*option_manager: flake8.options.manager.OptionManager*)
Add the offline option.

check_files(*present: bool*) → Iterator[Tuple[int, int, str, Type]]
Check files that should be present or absent.

error_base_number = 100

error_prefix = ''

flake8_error(*number: int*, *message: str*, *suggestion: str = None*, *add_to_base_number=True*) → Tuple[int, int, str, Type]
Return a flake8 error as a tuple.

name = 'nitpick'

static parse_options(*option_manager: flake8.options.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 = '0.22.2'

warn_missing_different(*comparison:* nitpick.formats.Comparison, *prefix_message: str = ''*)
Warn about missing and different keys.

nitpick.formats module

Configuration file formats.

class nitpick.formats.**BaseFormat**(***, *path: Union[pathlib.Path, str] = None*, *string: str = None*, *data: Union[Dict[str, Any], ruamel.yaml.comments.CommentedSeq, ruamel.yaml.comments.CommentedMap, BaseFormat] = None*, *ignore_keys: List[str] = None*)
Bases: `object`

Base class for configuration file formats.

Parameters

- **path** – Path of the config file to be loaded.
- **string** – Config in string format.
- **data** – Config data in Python format (dict, YamlFormat, TomlFormat instances).
- **ignore_keys** – List of keys to ignore when using the comparison methods.

property as_data
String content converted to a Python data structure (a dict, YAML data, etc.).

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

classmethod cleanup(**args: List[Any]*) → List[Any]
Cleanup similar values according to the specific format. E.g.: YamlFormat accepts 'True' or 'true'.

compare_with_dictdiffer(*expected: Union[Dict[str, Any], BaseFormat] = None*, *transform_function: Callable = None*) → *nitpick.formats.Comparison*
Compare two structures and compute missing and different items using `dictdiffer`.

compare_with_flatten(*expected: Union[Dict[str, Any], BaseFormat] = None*) → *nitpick.formats.Comparison*
Compare two flattened dictionaries and compute missing and different items.

abstract load()
Load the configuration from a file, a string or a dict.

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

class nitpick.formats.**Comparison**(*actual: Union[Dict[str, Any], ruamel.yaml.comments.CommentedSeq, ruamel.yaml.comments.CommentedMap, BaseFormat]*, *expected: Union[Dict[str, Any], ruamel.yaml.comments.CommentedSeq, ruamel.yaml.comments.CommentedMap, BaseFormat]*, *format_class: Type[BaseFormat]*)
Bases: `object`

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

set_diff(*diff_dict*)
Set the diff dict and corresponding format.

set_missing(*missing_dict*)
Set the missing dict and corresponding format.

update_pair(*key*, *raw_expected*)
Update a key on one of the comparison dicts, with its raw expected value.

class nitpick.formats.**JsonFormat**(***, *path: Union[pathlib.Path, str] = None*, *string: str = None*, *data: Union[Dict[str, Any], ruamel.yaml.comments.CommentedSeq, ruamel.yaml.comments.CommentedMap, BaseFormat] = None*, *ignore_keys: List[str] = None*)
Bases: *nitpick.formats.BaseFormat*

JSON configuration format.

property as_data
String content converted to a Python data structure (a dict, YAML data, etc.).

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

classmethod cleanup(**args: List[Any]*) → List[Any]
Cleanup similar values according to the specific format. E.g.: YamlFormat accepts 'True' or 'true'.

compare_with_dictdiffer(*expected: Union[Dict[str, Any], BaseFormat] = None*, *transform_function: Callable = None*) → *nitpick.formats.Comparison*
Compare two structures and compute missing and different items using `dictdiffer`.

compare_with_flatten(*expected: Union[Dict[str, Any], BaseFormat] = None*) → *nitpick.formats.Comparison*
Compare two flattened dictionaries and compute missing and different items.

load() → bool
Load a JSON file by its path, a string or a dict.

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

class nitpick.formats.**TomlFormat**(***, *path: Union[pathlib.Path, str] = None*, *string: str = None*, *data: Union[Dict[str, Any], ruamel.yaml.comments.CommentedSeq, ruamel.yaml.comments.CommentedMap, BaseFormat] = None*, *ignore_keys: List[str] = None*)
Bases: *nitpick.formats.BaseFormat*

TOML configuration format.

property as_data
String content converted to a Python data structure (a dict, YAML data, etc.).

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

classmethod cleanup(**args: List[Any]*) → List[Any]
Cleanup similar values according to the specific format. E.g.: YamlFormat accepts 'True' or 'true'.

compare_with_dictdiffer(*expected: Union[Dict[str, Any], BaseFormat] = None*, *transform_function: Callable = None*) → *nitpick.formats.Comparison*
Compare two structures and compute missing and different items using `dictdiffer`.

compare_with_flatten(*expected: Union[Dict[str, Any], BaseFormat] = None*) → *nitpick.formats.Comparison*
Compare two flattened dictionaries and compute missing and different items.

static group_name_for(*file_name: str*) → str
Return a TOML group name for a file name.

If the file name begins with a dot, remove it. Otherwise an error is raised: TomlDecodeError: Invalid group name 'xxx'. Try quoting it.

load() → bool
Load a TOML file by its path, a string or a dict.

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

class nitpick.formats.**YamlFormat**(***, *path: Union[pathlib.Path, str] = None*, *string: str = None*, *data: Union[Dict[str, Any], ruamel.yaml.comments.CommentedSeq, ruamel.yaml.comments.CommentedMap, BaseFormat] = None*, *ignore_keys: List[str] = None*)
Bases: *nitpick.formats.BaseFormat*

YAML configuration format.

property as_data
On YAML, this dict is a special object with comments and ordered keys.

property as_list
A list of dicts. On YAML, `as_data` might contain a `list`. This property is just a proxy for typing.

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

classmethod cleanup(**args: List[Any]*) → List[Any]
A boolean "True" or "true" might have the same effect on YAML.

compare_with_dictdiffer(*expected: Union[Dict[str, Any], BaseFormat] = None*, *transform_function: Callable = None*) → *nitpick.formats.Comparison*
Compare two structures and compute missing and different items using `dictdiffer`.

compare_with_flatten(*expected: Union[Dict[str, Any], BaseFormat] = None*) → *nitpick.formats.Comparison*
Compare two flattened dictionaries and compute missing and different items.

load() → bool
Load a YAML file by its path, a string or a dict.

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

nitpick.generic module

Generic functions and classes.

class nitpick.generic.**MergeDict**(*original_dict: Dict[str, Any] = None*)
Bases: object

A dictionary that can merge other dictionaries into it.

add(*other: Dict[str, Any]*) → None
Add another dictionary to the existing data.

merge() → Dict[str, Any]
Merge the dictionaries, replacing values with identical keys and extending lists.

nitpick.generic.**climb_directory_tree**(*starting_path: Union[pathlib.Path, str]*, *file_patterns: Iterable[str]*) → Optional[Set[pathlib.Path]]
Climb the directory tree looking for file patterns.

nitpick.generic.**find_object_by_key**(*list_: List[dict]*, *search_key: str*, *search_value: Any*) → dict
Find an object in a list, using a key/value pair to search.

```
>>> fruits = [{"id": 1, "fruit": "banana"}, {"id": 2, "fruit": "apple"}, {"id": 3,
↪ "fruit": "mango"}]
>>> find_object_by_key(fruits, "id", 1) == {'id': 1, 'fruit': 'banana'}
True
>>> find_object_by_key(fruits, "fruit", "banana") == {'id': 1, 'fruit': 'banana'}
True
>>> find_object_by_key(fruits, "fruit", "pear")
{}
>>> find_object_by_key(fruits, "fruit", "mango") == {'id': 3, 'fruit': 'mango'}
True
```

(continues on next page)

(continued from previous page)

```
>>> find_object_by_key(None, "fruit", "pear")
{}
```

nitpick.generic.**flatten**(*dict_*, *parent_key=''*, *separator='.'*, *current_lists=None*) → Dict[str, Any]

Flatten a nested dict.

Use *unflatten()* to revert.

Adapted from this StackOverflow question.

```
>>> expected = {'root.sub1': 1, 'root.sub2.deep': 3, 'sibling': False}
>>> flatten({"root": {"sub1": 1, "sub2": {"deep": 3}}, "sibling": False}) ==␣
↪expected
True
>>> expected = {'parent."with.dot".again': True, 'parent."my.my"': 1, "parent.123
↪": "numeric-key"}
>>> flatten({"parent": {"with.dot": {"again": True}, "my.my": 1, 123: "numeric-key
↪"}}) == expected
True
```

nitpick.generic.**get_subclasses**(*cls*)

Recursively get subclasses of a parent class.

nitpick.generic.**is_url**(*url: str*) → bool

Return True if a string is a URL.

```
>>> is_url("")
False
>>> is_url("  ")
False
>>> is_url("http://example.com")
True
```

nitpick.generic.**pretty_exception**(*err: Exception*, *message: str*)

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

nitpick.generic.**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.generic.**search_dict**(*jmespath_expression: Union[jmespath.parser.ParsedResult, str]*, *data: MutableMapping[str, Any]*, *default: Any*) → Any

Search a dictionary using a JMESPath expression, and returning a default value.

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

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

Parameters

- **jmespath_expression** – A compiled JMESPath expression or a string with an expression.
- **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.generic.**unflatten**(*dict_*, *separator='.'*) → collections.OrderedDict

Turn back a flattened dict created by *`flatten()`* into a nested dict.

```
>>> expected = {'my': {'sub': {'path': True}, 'home': 4}, 'another': {'path': 3}}
>>> unflatten({"my.sub.path": True, "another.path": 3, "my.home": 4}) == expected
True
>>> unflatten({"repo": "conflicted key", "repo.name": "?", "repo.path": "?"})
Traceback (most recent call last):
  ...
TypeError: 'str' object does not support item assignment
```

nitpick.generic.**version_to_tuple**(*version: str = None*) → Tuple[int, ...]

Transform a version number into a tuple of integers, for comparison.

```
>>> version_to_tuple("")
()
>>> version_to_tuple("  ")
()
>>> version_to_tuple(None)
()
>>> version_to_tuple("1.0.1")
(1, 0, 1)
>>> version_to_tuple(" 0.2 ")
(0, 2)
>>> version_to_tuple(" 2 ")
(2,)
```

Parameters **version** – String with the version number. It must be integers split by dots.

Returns Tuple with the version number.

nitpick.mixin module

Mixin to raise flake8 errors.

class nitpick.mixin.**NitpickMixin**
Bases: object

Mixin to raise flake8 errors.

error_base_number: int = 0

error_prefix: str = ''

flake8_error(*number: int*, *message: str*, *suggestion: str = None*, *add_to_base_number=True*) → Tuple[int, int, str, Type]
Return a flake8 error as a tuple.

warn_missing_different(*comparison:* nitpick.formats.Comparison, *prefix_message: str = ''*)
Warn about missing and different keys.

nitpick.schemas module

Marshmallow schemas.

class nitpick.schemas.**BaseNitpickSchema**(*, *only: Union[Sequence[str], Set[str]] = None*, *exclude: Union[Sequence[str], Set[str]] = ()*, *many: bool = False*, *context: Dict = None*, *load_only: Union[Sequence[str], Set[str]] = ()*, *dump_only: Union[Sequence[str], Set[str]] = ()*, *partial: Union[bool, Sequence[str], Set[str]] = False*, *unknown: str = None*)
Bases: marshmallow.schema.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.

- **render_module: Module to use for *loads <Schema.loads>* and *dumps <Schema.dumps>*.** Defaults to *json* from the standard library.
- **ordered: If *True*, order serialization output according to the** order in which fields were declared. 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 `marshmallow.schema.SchemaOpts`

TYPE_MAPPING = {<class 'str'>: <class 'marshmallow.fields.String'>, <class 'bytes'>:

property dict_class

dump(*obj: Any*, *, *many: bool = 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 A dict of serialized data

Return type dict

New in version 1.0.0.

Changed in version 3.0.0b7: This method returns the serialized data rather than a `(data, errors)` duple. 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*, ***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

Return type str

New in version 1.0.0.

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

error_messages = {'unknown': 'Unknown configuration. See https://nitpick.rtfd.io/en/l

classmethod from_dict(*fields: Dict[str, Union[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: marshmallow.exceptions.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: Union[Mapping[str, Any], Iterable[Mapping[str, Any]]]*, ***, *many: bool = None*, *partial: Union[bool, Sequence[str], Set[str]] = None*, *unknown: str = 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)` duple. A `ValidationError` is raised if invalid data are passed.

loads(*json_data: str*, ***, *many: bool = None*, *partial: Union[bool, Sequence[str], Set[str]] = None*, *unknown: str = 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)` duple. A `ValidationError` is raised if invalid data are passed.

on_bind_field(*field_name: str*, *field_obj: marshmallow.fields.Field*) → None
Hook to modify a field when it is bound to the *Schema*.

No-op by default.

opts = <marshmallow.schema.SchemaOpts object>

property set_class

validate(*data: Mapping*, ***, *many: bool = None*, *partial: Union[bool, Sequence[str], Set[str]] = 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: Union[Sequence[str], Set[str]] = None*, *exclude: Union[Sequence[str], Set[str]] = ()*, *many: bool = False*, *context: Dict = None*, *load_only: Union[Sequence[str], Set[str]] = ()*, *dump_only: Union[Sequence[str], Set[str]] = ()*, *partial: Union[bool, Sequence[str], Set[str]] = False*, *unknown: str = None*)
Bases: `marshmallow.schema.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.
- **render_module: Module to use for *loads* <*Schema.loads*> and *dumps* <*Schema.dumps*>.** Defaults to *json* from the standard library.
- **ordered: If *True*, order serialization output according to the** order in which fields were declared. 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 marshmallow.schema.SchemaOpts

TYPE_MAPPING = {<class 'str'>: <class 'marshmallow.fields.String'>, <class 'bytes'>:

property dict_class

dump(*obj: Any*, ***, *many: bool = 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 A dict of serialized data

Return type dict

New in version 1.0.0.

Changed in version 3.0.0b7: This method returns the serialized data rather than a `(data, errors)` duple. 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*, ***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

Return type str

New in version 1.0.0.

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

error_messages = {'unknown': 'Unknown file. See https://nitpick.rtfd.io/en/latest/con

classmethod from_dict(*fields: Dict[str, Union[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: marshmallow.exceptions.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: Union[Mapping[str, Any], Iterable[Mapping[str, Any]]]*, ***, *many: bool = None*, *partial: Union[bool, Sequence[str], Set[str]] = None*, *unknown: str = 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)` duple. A `ValidationError` is raised if invalid data are passed.

loads(*json_data: str*, ***, *many: bool = None*, *partial: Union[bool, Sequence[str], Set[str]] = None*, *unknown: str = 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)` duple. A `ValidationError` is raised if invalid data are passed.

on_bind_field(*field_name: str*, *field_obj: marshmallow.fields.Field*) → None
Hook to modify a field when it is bound to the *Schema*.

No-op by default.

opts = <marshmallow.schema.SchemaOpts object>

property set_class

validate(*data: Mapping*, ***, *many: bool = None*, *partial: Union[bool, Sequence[str], Set[str]] = 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: Union[Sequence[str], Set[str]] = None*, *exclude: Union[Sequence[str], Set[str]] = ()*, *many: bool = False*, *context: Dict = None*, *load_only: Union[Sequence[str], Set[str]] = ()*, *dump_only: Union[Sequence[str], Set[str]] = ()*, *partial: Union[bool, Sequence[str], Set[str]] = False*, *unknown: str = None*)

Bases: *nitpick.schemas.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.
- **render_module: Module to use for *loads <Schema.loads>* and *dumps <Schema.dumps>*.** Defaults to *json* from the standard library.
- **ordered: If *True*, order serialization output according to the** order in which fields were declared. 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 `marshmallow.schema.SchemaOpts`

TYPE_MAPPING = {<class 'str'>: <class 'marshmallow.fields.String'>, <class 'bytes'>:

property dict_class

dump(*obj: Any*, *, *many: bool = 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 A dict of serialized data

Return type dict

New in version 1.0.0.

Changed in version 3.0.0b7: This method returns the serialized data rather than a `(data, errors)` duple. 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*, ***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

Return type str

New in version 1.0.0.

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

error_messages = {'unknown': 'Unknown file. See https://nitpick.rtfd.io/en/latest/nitp

classmethod from_dict(*fields: Dict[str, Union[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: marshmallow.exceptions.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: Union[Mapping[str, Any], Iterable[Mapping[str, Any]]]*, ***, *many: bool = None*, *partial: Union[bool, Sequence[str], Set[str]] = None*, *unknown: str = 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)` duple. A `ValidationError` is raised if invalid data are passed.

loads(*json_data: str*, ***, *many: bool = None*, *partial: Union[bool, Sequence[str], Set[str]] = None*, *unknown: str = 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)` duple. A `ValidationError` is raised if invalid data are passed.

on_bind_field(*field_name: str*, *field_obj: marshmallow.fields.Field*) → None
Hook to modify a field when it is bound to the *Schema*.

No-op by default.

opts = <marshmallow.schema.SchemaOpts object>

property set_class

validate(*data: Mapping*, ***, *many: bool = None*, *partial: Union[bool, Sequence[str], Set[str]] = 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.**NitpickJSONFileSectionSchema**(***, *only: Union[Sequence[str], Set[str]] = None*, *exclude: Union[Sequence[str], Set[str]] = ()*, *many: bool = False*, *context: Dict = None*, *load_only: Union[Sequence[str], Set[str]] = ()*, *dump_only: Union[Sequence[str], Set[str]] = ()*, *partial: Union[bool, Sequence[str], Set[str]] = False*, *unknown: str = None*)
Bases: *nitpick.schemas.BaseNitpickSchema*

Validation schema for the `[nitpick.JSONFile]` 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.
- **`render_module`: Module to use for *loads* <*Schema.loads*> and *dumps* <*Schema.dumps*>.** Defaults to *json* from the standard library.
- **`ordered`: If *True*, order serialization output according to the** order in which fields were declared. 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 `marshmallow.schema.SchemaOpts`

`TYPE_MAPPING = {<class 'str'>: <class 'marshmallow.fields.String'>, <class 'bytes'>:`

`property dict_class`

`dump`(*obj: Any*, *, *many: bool = 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 A dict of serialized data

Return type dict

New in version 1.0.0.

Changed in version 3.0.0b7: This method returns the serialized data rather than a `(data, errors)` duple. 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*, ***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

Return type str

New in version 1.0.0.

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

error_messages = {'unknown': 'Unknown configuration. See https://nitpick.rtfd.io/en/l

classmethod from_dict(*fields: Dict[str, Union[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: marshmallow.exceptions.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: Union[Mapping[str, Any], Iterable[Mapping[str, Any]]]*, ***, *many: bool = None*, *partial: Union[bool, Sequence[str], Set[str]] = None*, *unknown: str = 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)` duple. A `ValidationError` is raised if invalid data are passed.

loads(*json_data: str*, ***, *many: bool = None*, *partial: Union[bool, Sequence[str], Set[str]] = None*, *unknown: str = 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)` duple. A `ValidationError` is raised if invalid data are passed.

on_bind_field(*field_name: str*, *field_obj: marshmallow.fields.Field*) → None
Hook to modify a field when it is bound to the *Schema*.

No-op by default.

opts = <marshmallow.schema.SchemaOpts object>

property set_class

validate(*data: Mapping*, ***, *many: bool = None*, *partial: Union[bool, Sequence[str], Set[str]] = 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: Union[Sequence[str], Set[str]] = None*, *exclude: Union[Sequence[str], Set[str]] = ()*, *many: bool = False*, *context: Dict = None*, *load_only: Union[Sequence[str], Set[str]] = ()*, *dump_only: Union[Sequence[str], Set[str]] = ()*, *partial: Union[bool, Sequence[str], Set[str]] = False*, *unknown: str = None*)

Bases: *nitpick.schemas.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.
- **render_module: Module to use for *loads <Schema.loads>* and *dumps <Schema.dumps>*.** Defaults to *json* from the standard library.
- **ordered: If *True*, order serialization output according to the** order in which fields were declared. 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 marshmallow.schema.SchemaOpts

TYPE_MAPPING = {<class 'str'>: <class 'marshmallow.fields.String'>, <class 'bytes'>:

property dict_class

dump(*obj: Any*, *, *many: bool = 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 A dict of serialized data

Return type dict

New in version 1.0.0.

Changed in version 3.0.0b7: This method returns the serialized data rather than a `(data, errors)` duple. 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*, ***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

Return type str

New in version 1.0.0.

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

error_messages = {'unknown': 'Unknown configuration. See https://nitpick.rtfd.io/en/l

classmethod from_dict(*fields: Dict[str, Union[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: marshmallow.exceptions.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: Union[Mapping[str, Any], Iterable[Mapping[str, Any]]]*, ***, *many: bool = None*, *partial: Union[bool, Sequence[str], Set[str]] = None*, *unknown: str = 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)` duple. A `ValidationError` is raised if invalid data are passed.

loads(*json_data: str*, ***, *many: bool = None*, *partial: Union[bool, Sequence[str], Set[str]] = None*, *unknown: str = 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)` duple. A `ValidationError` is raised if invalid data are passed.

on_bind_field(*field_name: str*, *field_obj: marshmallow.fields.Field*) → None
Hook to modify a field when it is bound to the *Schema*.

No-op by default.

opts = <marshmallow.schema.SchemaOpts object>

property set_class

validate(*data: Mapping*, ***, *many: bool = None*, *partial: Union[bool, Sequence[str], Set[str]] = 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: Union[Sequence[str], Set[str]] = None*, *exclude: Union[Sequence[str], Set[str]] = ()*, *many: bool = False*, *context: Dict = None*, *load_only: Union[Sequence[str], Set[str]] = ()*, *dump_only: Union[Sequence[str], Set[str]] = ()*, *partial: Union[bool, Sequence[str], Set[str]] = False*, *unknown: str = None*)

Bases: *nitpick.schemas.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.
- **render_module: Module to use for *loads <Schema.loads>* and *dumps <Schema.dumps>*.** Defaults to *json* from the standard library.
- **ordered: If *True*, order serialization output according to the** order in which fields were declared. 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 `marshmallow.schema.SchemaOpts`

TYPE_MAPPING = {<class 'str'>: <class 'marshmallow.fields.String'>, <class 'bytes'>:

property dict_class

dump(*obj: Any*, ***, *many: bool = 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 A dict of serialized data

Return type dict

New in version 1.0.0.

Changed in version 3.0.0b7: This method returns the serialized data rather than a `(data, errors)` duple. 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*, ***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

Return type str

New in version 1.0.0.

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

error_messages = {'unknown': 'Unknown configuration. See https://nitpick.rtfd.io/en/l

classmethod from_dict(*fields: Dict[str, Union[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: marshmallow.exceptions.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: Union[Mapping[str, Any], Iterable[Mapping[str, Any]]]*, ***, *many: bool = None*, *partial: Union[bool, Sequence[str], Set[str]] = None*, *unknown: str = 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)` duple. A `ValidationError` is raised if invalid data are passed.

loads(*json_data: str*, ***, *many: bool = None*, *partial: Union[bool, Sequence[str], Set[str]] = None*, *unknown: str = 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)` duple. A `ValidationError` is raised if invalid data are passed.

on_bind_field(*field_name: str*, *field_obj: marshmallow.fields.Field*) → None
Hook to modify a field when it is bound to the *Schema*.

No-op by default.

opts = <marshmallow.schema.SchemaOpts object>

property set_class

validate(*data: Mapping*, ***, *many: bool = None*, *partial: Union[bool, Sequence[str], Set[str]] = 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.**SetupCfgSchema**(*, *only: Union[Sequence[str], Set[str]] = None*, *exclude: Union[Sequence[str], Set[str]] = ()*, *many: bool = False*, *context: Dict = None*, *load_only: Union[Sequence[str], Set[str]] = ()*, *dump_only: Union[Sequence[str], Set[str]] = ()*, *partial: Union[bool, Sequence[str], Set[str]] = False*, *unknown: str = None*)

Bases: *nitpick.schemas.BaseNitpickSchema*

Validation schema for setup.cfg.

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.
- **render_module: Module to use for *loads <Schema.loads>* and *dumps <Schema.dumps>*.** Defaults to *json* from the standard library.
- **ordered: If *True*, order serialization output according to the** order in which fields were declared. 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 marshmallow.schema.SchemaOpts

TYPE_MAPPING = {<class 'str'>: <class 'marshmallow.fields.String'>, <class 'bytes'>:

property dict_class

dump(*obj: Any*, *, *many: bool = 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 A dict of serialized data

Return type dict

New in version 1.0.0.

Changed in version 3.0.0b7: This method returns the serialized data rather than a `(data, errors)` duple. 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*, ***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

Return type str

New in version 1.0.0.

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

error_messages = {'unknown': 'Unknown configuration. See https://nitpick.rtfd.io/en/l

classmethod from_dict(*fields: Dict[str, Union[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: marshmallow.exceptions.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: Union[Mapping[str, Any], Iterable[Mapping[str, Any]]]*, *, *many: bool = None*, *partial: Union[bool, Sequence[str], Set[str]] = None*, *unknown: str = 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)` duple. A `ValidationError` is raised if invalid data are passed.

loads(*json_data: str*, *, *many: bool = None*, *partial: Union[bool, Sequence[str], Set[str]] = None*, *unknown: str = 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)` duple. A `ValidationError` is raised if invalid data are passed.

on_bind_field(*field_name: str*, *field_obj: marshmallow.fields.Field*) → None
Hook to modify a field when it is bound to the *Schema*.

No-op by default.

opts = <marshmallow.schema.SchemaOpts object>

property set_class

validate(*data: Mapping*, ***, *many: bool = None*, *partial: Union[bool, Sequence[str], Set[str]] = 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.**ToolNitpickSectionSchema**(***, *only: Union[Sequence[str], Set[str]] = None*, *exclude: Union[Sequence[str], Set[str]] = ()*, *many: bool = False*, *context: Dict = None*, *load_only: Union[Sequence[str], Set[str]] = ()*, *dump_only: Union[Sequence[str], Set[str]] = ()*, *partial: Union[bool, Sequence[str], Set[str]] = False*, *unknown: str = None*)
Bases: *nitpick.schemas.BaseNitpickSchema*

Validation schema for the `[tool.nitpick]` section 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.
- **`render_module`: Module to use for *loads <Schema.loads>* and *dumps <Schema.dumps>*.** Defaults to *json* from the standard library.
- **`ordered`: If *True*, order serialization output according to the** order in which fields were declared. 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 `marshmallow.schema.SchemaOpts`

`TYPE_MAPPING = {<class 'str'>: <class 'marshmallow.fields.String'>, <class 'bytes'>:`

`property dict_class`

`dump`(*obj: Any*, *, *many: bool = 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 A dict of serialized data

Return type dict

New in version 1.0.0.

Changed in version 3.0.0b7: This method returns the serialized data rather than a `(data, errors)` duple. 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*, ***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

Return type str

New in version 1.0.0.

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

error_messages = {'unknown': 'Unknown configuration. See https://nitpick.rtfd.io/en/l

classmethod from_dict(*fields: Dict[str, Union[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: marshmallow.exceptions.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: Union[Mapping[str, Any], Iterable[Mapping[str, Any]]]*, ***, *many: bool = None*, *partial: Union[bool, Sequence[str], Set[str]] = None*, *unknown: str = 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)` duple. A `ValidationError` is raised if invalid data are passed.

loads(*json_data: str*, ***, *many: bool = None*, *partial: Union[bool, Sequence[str], Set[str]] = None*, *unknown: str = 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)` duple. A `ValidationError` is raised if invalid data are passed.

on_bind_field(*field_name: str*, *field_obj: marshmallow.fields.Field*) → None
Hook to modify a field when it is bound to the *Schema*.

No-op by default.

opts = <marshmallow.schema.SchemaOpts object>

property set_class

validate(*data: Mapping*, ***, *many: bool = None*, *partial: Union[bool, Sequence[str], Set[str]] = 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 files.

class nitpick.style.**Style**
Bases: object

Include styles recursively from one another.

fetch_style_from_local_path(*partial_file_name: str*) → Optional[pathlib.Path]
Fetch a style file from a local path.

fetch_style_from_url(*url: str*) → Optional[pathlib.Path]
Fetch a style file from a URL, saving the contents in the cache dir.

static file_field_pair(*file_name: str*, *base_file_class: Type[*nitpick.plugins.base.BaseFile*]*) → Dict[str, marshmallow.fields.Field]
Return a schema field with info from a config file class.

find_initial_styles(*configured_styles: Union[str, List[str]]*)
Find the initial style(s) and include them.

static get_default_style_url()
Return the URL of the default style for the current version.

get_style_path(*style_uri: str*) → Optional[pathlib.Path]
Get the style path from the URI. Add the .toml extension if it's missing.

include_multiple_styles(*chosen_styles: Union[str, List[str]]*) → None
Include a list of styles (or just one) into this style tree.

merge_toml_dict() → Dict[str, Any]
Merge all included styles into a TOML (actually JSON) dictionary.

rebuild_dynamic_schema(*data: Dict[str, Any] = None*) → None
Rebuild the dynamic Marshmallow schema when needed, adding new fields that were found on the style.

validate_style(*style_file_name: str*, *original_data: Dict[str, Any]*)
Validate a style file (TOML) against a Marshmallow schema.

nitpick.typedefs module

Type definitions.

CHAPTER

TWELVE

INDICES AND TABLES

- genindex
- modindex
- search

CHAPTER

THIRTEEN

TO DO LIST

PYTHON MODULE INDEX

n

INDEX

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

Y