# Simple Logging Documentation

***Release 0.10.0***

**Vincent Poulailleau**

**Sep 16, 2019**

# Contents:

CHAPTER 1

# Simple Logging

PyPI PyPI Travis ReadTheDocs

Logging made simple, no excuse for any debug print call.

- Free software: BSD 3-Clause license
- Documentation: https://simplelogging.readthedocs.io.

## 1.1 Features

- Logging management (debug, information or error messages)
- Simple logging setup
- Based on Python `logging` module of the standard library
- Based on colorlog for colored log on console

For advanced users:

- The provided logger is one of those from `logging`, this means it can be configured so that log messages are sent by email, HTTP, or any of the options available in https://docs.python.org/3/library/logging.handlers.html.
- The StreamHandler and the associated Formatter are those provided by `colorlog`.

## 1.2 Example

### 1.2.1 Basic usage

```
import simplelogging

# log = simplelogging.get_logger(console_level=simplelogging.DEBUG)
```

(continues on next page)

(continued from previous page)

```
# log = simplelogging.get_logger(file_name="log.txt")
log = simplelogging.get_logger()

a_string_variable = "hello"
an_integer_variable = 42
a_floating_point_variable = 3.14

log.debug("some debug")
log.info("some info")
log.info(
    "some variables: %s, %d, %f",
    a_string_variable,
    an_integer_variable,
    a_floating_point_variable,
)
log.warning("some warning")
log.error("some error")
log.critical("some critical error")

try:
    x = 1 / 0
except ZeroDivisionError as error:
    log.exception(error)
```

```
2018-12-09 16:16:46,656 [INFO    ]              main.py( 12):<module>             :: some info
2018-12-09 16:16:46,657 [INFO    ]              main.py( 17):<module>             :: some variables: hello, 42, 3.140000
2018-12-09 16:16:46,657 [WARNING ]              main.py( 19):<module>             :: some warning
2018-12-09 16:16:46,657 [ERROR   ]              main.py( 20):<module>             :: some error
2018-12-09 16:16:46,658 [CRITICAL]              main.py( 21):<module>             :: some critical error
2018-12-09 16:16:46,658 [ERROR   ]              main.py( 26):<module>             :: division by zero
Traceback (most recent call last):
  File "/home/vincent/documents/programmation/simplelogging/example/quick start/main.py", line 24, in <module>
    x = 1 / 0
ZeroDivisionError: division by zero
```

quickstart result

Keep in mind that you shouldn't do string formatting yourself. Delegate formatting to `simplelogging` (i.e. `logging` in this case), the formatting will be done only if necessary, that is if the message is going to be displayed. See above examples of how to display variables.

### 1.2.2 Usage with modules

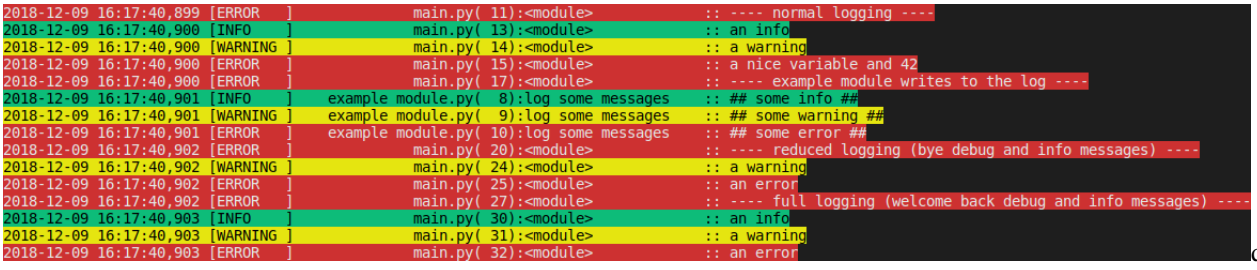
#### example_module.py

```
import simplelogging

log = simplelogging.get_logger()


def log_some_messages():
    log.debug("## some debug ##")
    log.info("## some info ##")
    log.warning("## some warning ##")
    log.error("## some error ##")
```

### main.py

```
import example_module
import simplelogging

# log = simplelogging.get_logger(console_level=simplelogging.DEBUG)
# log = simplelogging.get_logger(file_name="log.txt")
log = simplelogging.get_logger()

a_variable = "a nice variable"
another_variable = 42

log.error("---- normal logging ----")
log.debug("a debug message")
log.info("an info")
log.warning("a warning")
log.error("%s and %d", a_variable, another_variable)

log.error("---- example_module writes to the log ----")
example_module.log_some_messages()

log.error("---- reduced logging (bye debug and info messages) ----")
log.reduced_logging()
log.debug("a debug message")
log.info("an info")
log.warning("a warning")
log.error("an error")

log.error("---- full logging (welcome back debug and info messages) ----")
log.full_logging()
log.debug("a debug message")
log.info("an info")
log.warning("a warning")
log.error("an error")
```

### Result in the console



quickstart with modules result

More examples are provided in the documentation: https://simplelogging.readthedocs.io.

## 1.3 TODO

- add tests
- add type annotations

- add docstring
- commit hooks
- describe pros/cons and alternatives
- release 1.0!

## 1.4 Credits

This package is an extension of the logging package in the Python standard library. Coloring of the console relies on colorlog.

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

CHAPTER 2

# Installation

## 2.1 Stable release

To install Simple Logging, run this command in your terminal:

```
$ pip install simplelogging
```

This is the preferred method to install Simple Logging, as it will always install the most recent stable release.

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

## 2.2 From sources

The sources for Simple Logging can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/vpoulailleau/simplelogging
```

Or download the tarball:

```
$ curl  -OL https://github.com/vpoulailleau/simplelogging/tarball/master
```

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

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

CHAPTER 3

# Usage

## 3.1 Quick start

To use Simple Logging in a project

```
import simplelogging

# log = simplelogging.get_logger(console_level=simplelogging.DEBUG)
# log = simplelogging.get_logger(file_name="log.txt")
log = simplelogging.get_logger()

a_string_variable = "hello"
an_integer_variable = 42
a_floating_point_variable = 3.14

log.debug("some debug")
log.info("some info")
log.info(
    "some variables: %s, %d, %f",
    a_string_variable,
    an_integer_variable,
    a_floating_point_variable,
)
log.warning("some warning")
log.error("some error")
log.critical("some critical error")

try:
    x = 1 / 0
except ZeroDivisionError as error:
    log.exception(error)
```

Note: `log` variable is a logger from the Python standard library's `logging` module.

## 3.2 Configure the logger

The logger provided by `simplelogging.get_logger()` is already configured to be ready to use:

- Console output: info, warning, error and critical messages displayed. Colored output.
- No file output

The only logger that is configured is the one you have in your main script (i.e. not in an imported module). For imported module, log messages will be forwarded to the main logger.

To get a not yet configured logger, you can use:

```
import simplelogging

log = simplelogging.get_logger(console=False)
```

### 3.2.1 Console output

The console output is managed through a `colorlog.StreamHandler` configured by `simplelogging`.

#### Disabling the console output

As already said, you can avoid having a console output by doing this in your main script:

```
import simplelogging

log = simplelogging.get_logger(console=False)
```

#### Changing message format on console

```
import simplelogging

log = simplelogging.get_logger(
    console_format="%(asctime)s")
```

You can configure de message format according to https://docs.python.org/3/library/logging.html#logrecord-attributes and https://github.com/borntyping/python-colorlog.

#### Changing message level on console

```
import simplelogging

log = simplelogging.get_logger(
    console_level=simplelogging.DEBUG)
```

The logger will display on the console only messages with the level set to provide value or above.

For example, the above code allows debug, info, warning and error messages to be displayed in the console.

`simplelogging.DEBUG` is `logging.DEBUG`, and same for `INFO`, `WARNING`, `ERROR`, `CRITICAL`. They are provided for convenience, avoiding to import `logging`.

See https://docs.python.org/3/library/logging.html#logging-levels and https://docs.python.org/3/howto/logging.html#when-to-use-logging for more detail.

### 3.2.2 File output

The file output is managed through a `logging.handlers.RotatingFileHandler` configured by `simplelogging`.

#### Disabling and enabling the file output

File output is disabled by default. But you can enable logging to a file by giving the file path in your main script:

```
import simplelogging

log = simplelogging.get_logger(
    file_name="log.txt")
```

#### Changing message format in the file

```
import simplelogging

log = simplelogging.get_logger(
    file_format="%(asctime)s")
```

You can configure de message format according to https://docs.python.org/3/library/logging.html#logrecord-attributes.

#### Changing message level in the file

```
import simplelogging

log = simplelogging.get_logger(
    file_level=simplelogging.DEBUG)
```

See above the explanations for console level, they are applicable for file level.

### 3.2.3 Configuring logger level

The logger level applies to both console and file output. Since the logger is a standard `logging.Logger`, you can use the `setLevel` method.

During the initial configuration, you can provide the logger level.

```
import simplelogging

log = simplelogging.get_logger(
    logger_level=simplelogging.DEBUG)
```

### 3.2.4 Configuring an existing logger

The logger can be configured directly with `logging` API. The only helpers provided by `simplelogging` are:

```
log = simplelogging.get_logger()

log.reduced_logging()
log.normal_logging()
log.full_logging()
```

Those three methods configure respectively the logger level to:

- `WARNING`
- `INFO`
- `DEBUG`

## 3.3 Default configuration

`simplelogging.get_logger()` is an easy way to configure a logging infrastructure. It accepts several parameters:

- name: name of the logger (default: `None`, `simplelogging` will call `logging.getLogger(__name__)` for imported modules)
- logger_level: logging level (default: `DEBUG`)
- console: activation of console output (default: `True`)
- console_format: message format on console (default: `DEFAULT_CONSOLE_FORMAT`)
- console_level: logging level of the console (default: `INFO`)
- file_name: name of the file in which the log will be written (default: `None`, i.e. no file)
- file_format: message format in the file (default: `DEFAULT_FILE_FORMAT`)
- file_level: logging level in the file (default: `DEBUG`)

Default formats are:

```
DEFAULT_CONSOLE_FORMAT = (
    "%(log_color)s%(asctime)s [%(levelname)-8s] "
    "%(filename)20s(%(lineno)3s):%(funcName)-20s ::"
    " %(message)s%(reset)s"
)

DEFAULT_FILE_FORMAT = (
    "%(asctime)s [%(levelname)-8s] "
    "%(filename)20s(%(lineno)3s):%(funcName)-20s ::"
    " %(message)s"
)
```

CHAPTER 4

# Contributing

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

You can contribute in many ways:

## 4.1 Types of Contributions

### 4.1.1 Report Bugs

Report bugs at https://github.com/vpoulailleau/simplelogging/issues.

If you are reporting a bug, please include:

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

### 4.1.2 Fix Bugs

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

### 4.1.3 Implement Features

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

### 4.1.4 Write Documentation

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

### 4.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/vpoulailleau/simplelogging/issues.

If you are proposing a feature:

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

## 4.2 Get Started!

Ready to contribute? Here's how to set up *simplelogging* for local development.

1. Fork the *simplelogging* repo on GitHub.
2. Clone your fork locally:

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

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

```
$ mkvirtualenv simplelogging
$ cd simplelogging/
$ python setup.py develop
```

4. Create a branch for local development:

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

   Now you can make your changes locally.

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

```
$ flake8 simplelogging tests
$ python setup.py test or py.test
$ tox
```

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

6. 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
```

7. Submit a pull request through the GitHub website.

## 4.3 Pull Request Guidelines

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

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 3.4, 3.5, 3.6 and 3.7, and for PyPy. Check https://travis-ci.org/vpoulailleau/simplelogging/pull_requests and make sure that the tests pass for all supported Python versions.

## 4.4 Tips

To run a subset of tests:

```
$ py.test tests.test_simplelogging
```

## 4.5 Deploying

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

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

Travis will then deploy to PyPI if tests pass.

CHAPTER 5

# Credits

## 5.1 Development Lead

- Vincent Poulailleau <vpoulailleau@gmail.com>

## 5.2 Contributors

None yet. Why not be the first?

CHAPTER 6

# History

## 6.1 0.10.0 (2019-09-16)

- setup.py: require pytest-runner only when necessary
- remove Python 3.4 support

## 6.2 0.9.0 (2018-12-14)

- Improve documentation
- Add tests
- Change API for easy logging level change

## 6.3 0.8.0 (2018-12-09)

- Improve documentation
- Change default format: enlarge level size for critical errors

## 6.4 0.7.0 (2018-12-08)

- Fix logging to file

## 6.5 0.6.0 (2018-12-07)

- Colored output on console

- Improved documentation

## 6.6 0.5.0 (2018-12-02)

- Fix README rendering in PyPI

## 6.7 0.4.0 (2018-12-02)

- Fix bump config

## 6.8 0.3.0 (2018-12-02)

- First release on PyPI.

CHAPTER 7

# Indices and tables

- genindex
- modindex
- search