neatlog

Release 0.1.8

Steffen Brinkmann

Feb 21, 2021

CONTENTS:

This package provides an easy and transparent way of customizing the builtin logging. Just import the module and enjoy the difference.

Table of Contents

CHAPTER

ONE

QUICK START

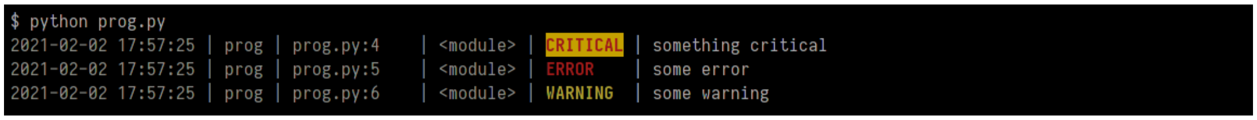
Install by typing

```
pip install neatlog
```

and simply import neatlog in your program and use logging as usually:

```python
import logging
import neatlog

logging.critical("something critical")
logging.error("some error")
logging.warning("some warning")
```



```
$ python prog.py
2021-02-02 17:57:25 | prog | prog.py:4    | <module> | CRITICAL | something critical
2021-02-02 17:57:25 | prog | prog.py:5    | <module> | ERROR    | some error
2021-02-02 17:57:25 | prog | prog.py:6    | <module> | WARNING  | some warning
```

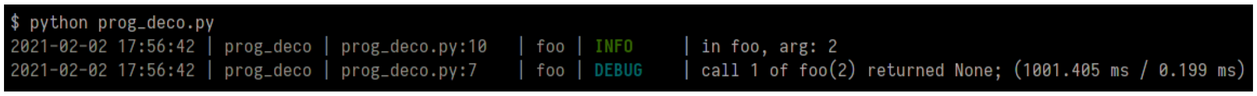
Or use the `log` decorator to log function calls:

```python
import logging
from time import sleep
from neatlog import log, set_log_level

set_log_level(logging.DEBUG)

@log
def foo(x):
    sleep(1)
    logging.info(f"in foo, arg: {x}")

foo(2)
```



```
$ python prog_deco.py
2021-02-02 17:56:42 | prog_deco | prog_deco.py:10   | foo | INFO     | in foo, arg: 2
2021-02-02 17:56:42 | prog_deco | prog_deco.py:7    | foo | DEBUG    | call 1 of foo(2) returned None; (1001.405 ms / 0.199 ms)
```

CHAPTER

TWO

WHY NOT USE THE BUILT IN LOGGING?

In fact, neatlog uses Python's built in logging, so let's rephrase the question: **"Why use neatlog on top of logging?"**

If you have ever found yourself looking up how to configure the logging format for the 195th time in order to get a more appealing or more parsable output, or if copying the `basic_config` command from another project over to the new one, then you may appreciate to get a neat and very readable logging setup by simply importing neatlog.

At the same time, neatlog is totally transparent, meaning it merely provides, configures, adds and removes handlers and formatters. So you can combine neatlog's configuration with your own handlers or access the handlers and formatters to fiddle around with them.

CHAPTER THREE

DOCUMENTATION

3.1 Installation

The installation is straight forward. You can install the package via `pip`, `pipenv`, `poetry` and alike or by downloading the source from the gitlab repository.

3.1.1 From pypi.org (recommended)

Install by typing

```
pip install neatlog
```

or

```
pip install --user neatlog
```

if you do not have root access.

Please check the documentations for pipenv, and poetry for information on how to install packages with these tools.

3.1.2 From gitlab.com (for experts)

To get the latest features or contribute to the development, you can clone the whole project using git:

```
git clone https://gitlab.com/szs/neatlog.git
```

3.2 Usage

Simply import neatlog in your program and use logging as usually:

```python
import logging
from log_fmt import set_log_level

set_log_level(logging.DEBUG)
logging.critical("something critical")
logging.error("some error")
logging.warning("some warning")
logging.info("some info")
logging.debug("something for debugging")
```

3.3 Configuration

tbd.

```
import logging
from log_fmt import set_log_level

set_log_level(logging.DEBUG)
logging.critical("something critical")
logging.error("some error")
logging.warning("some warning")
logging.info("some info")
logging.debug("something for debugging")
```

```
import logging
from log_fmt import config_logger

# switch on logging debug info to a file
config_logger(log_to_file=True, filename="debug.log", level=logging.DEBUG)

logging.critical("something critical")
logging.error("some error")
logging.warning("some warning")
logging.info("some info")
logging.debug("something for debugging")
```

CHAPTER

FOUR

HOW TO CONTRIBUTE

If you find a bug, want to propose a feature or need help getting this package to work with your data on your system, please don't hesitate to file an issue or write an email. Merge requests are also much appreciated!

CHAPTER

FIVE

PROJECT LINKS

- Repository
- Documentation
- pypi page