Pylava Documentation

Release 0.3.0

Pylava Developers

Sep 28, 2020

Contents

Pylava is a community maintained fork of Pylama.

Pylava is a code audit tool for Python and JavaScript. Pylava wraps these tools:

- pycodestyle (formerly pep8) © 2012-2013, Florent Xicluna;
- pydocstyle (formerly pep257 by Vladimir Keleshev) © 2014, Amir Rachum;
- PyFlakes © 2005-2013, Kevin Watters;
- Mccabe © Ned Batchelder;
- Pylint © 2013, Logilab (should be installed 'pylama_pylint' module);
- Radon © Michele Lacchia
- gjslint © The Closure Linter Authors (should be installed 'pylama_gjslint' module);

Contents

CHAPTER 1

Credits

Thanks to:

- Kirill Klenov for creating and maintaining the original Pylama project. This fork named Pylava is a derivative work based on Kirill Klenov's Pylama project.
- Contributors to Pylama.
- Contributors to Pylava.

CHAPTER 2

New in Pylava

This fork of Pylama differs from the original Pylama project in the following areas:

- Pylama does not work with Python 3.7 due to Pylama issue #123. While there is a pull request to resolve the issue, they are not being merged into the project due to lack of maintenance. This fork named Pylava is meant for merging useful pull requests into the project, so that the project can satsify the current needs of Python developers. This is the primary reason why this fork was created.
- The licensing terms of Pylama are unclear. The README of the original Pylama project mentioned:

  > Licensed under a BSD license.

  It is unclear which BSD license (BSD-3-Clause or BSD-2-Clause) is meant here. Moreover there are references to the GNU Lesser General Public License (GNU LGPL) also in the project. See Pylama issue #64 for more about this.

  This fork interprets the license section of the README to mean that the Pylama project is available under a BSD license in addition to certain files being available under GNU LGPL due to the mentions of GNU LGPL in such files.

  Further, this fork named Pylava (a derivative work based on Pylama) is distributed under the terms of the MIT license which is allowed by BSD licenses.
- While the original Pylama project uses the `develop` branch as the active development branch, this fork uses the `master` branch as the active development branch.

CHAPTER 3

Documentation

Documentation is available at https://pylavadocs.readthedocs.io/. Pull requests with documentation enhancements and/or fixes are awesome and most welcome.

CHAPTER 4

Requirements

- Python (2.7, 3.5, 3.6, 3.7, 3.8, or 3.9)
- To use JavaScript checker (`gjslint`) you need to install `python-gflags` with `pip install python-gflags`.
- If your tests are failing on Win platform you are missing: `curses` - http://www.lfd.uci.edu/~gohlke/pythonlibs/ (The curses library supplies a terminal-independent screen-painting and keyboard-handling facility for text-based terminals)

CHAPTER 5

Installation

Enter the following command to install Pylava.

```
$ pip install pylava
```

With Python 3, you may need to enter the following command instead.

```
$ pip3 install pylava
```

CHAPTER 6

Quick Start

Pylava is easy to use and really fun for checking code quality. Just run *pylava* and get common output from all pylava plugins (pycodestyle, PyFlakes and etc)

Recursively check the current directory.

```
$ pylava
```

Recursively check a path.

```
$ pylava <path_to_directory_or_file>
```

Ignore errors

```
$ pylava -i W,E501
```

Note: You could choose a group of errors `D`, `E1`, etc., or special errors `C0312`.

Choose code checkers

```
$ pylava -l "pycodestyle,mccabe"
```

Choose code checkers for JavaScript:

```
$ pylava --linters=gjslint --ignore=E:0010 <path_to_directory_or_file>
```

CHAPTER 7

Set Pylava (checkers) options

7.1 Command line options

```
$ pylava --help

usage: pylava [-h] [--verbose] [--version] [--format {pycodestyle,pylint}]
              [--select SELECT] [--sort SORT] [--linters LINTERS]
              [--ignore IGNORE] [--skip SKIP] [--report REPORT] [--hook]
              [--async] [--options OPTIONS] [--force] [--abspath]
              [paths [paths ...]]

Code audit tool for python.

positional arguments:
  paths                 Paths to files or directories for code check.

optional arguments:
  -h, --help            show this help message and exit
  --verbose, -v         Verbose mode.
  --version             show program's version number and exit
  --format {pycodestyle,pylint}, -f {pycodestyle,pylint}
                        Choose errors format (pycodestyle, pylint).
  --select SELECT, -s SELECT
                        Select errors and warnings. (comma-separated list)
  --sort SORT           Sort result by error types. Ex. E,W,D
  --linters LINTERS, -l LINTERS
                        Select linters. (comma-separated). Choices are
                        mccabe,pycodestyle,pyflakes,pydocstyle.
  --ignore IGNORE, -i IGNORE
                        Ignore errors and warnings. (comma-separated)
  --skip SKIP           Skip files by masks (comma-separated, Ex.
                        */messages.py)
  --report REPORT, -r REPORT
                        Send report to file [REPORT]
```

(continues on next page)

(continued from previous page)

```
  --hook                Install Git (Mercurial) hook.
  --async               Enable async mode. Useful for checking a lot of
                        files. Not supported by pylint.
  --options FILE, -o FILE
                        Specify configuration file. Looks for pylava.ini,
                        setup.cfg, tox.ini, or pytest.ini in the current
                        directory.
  --force, -F           Force code checking (if linter doesnt allow)
  --abspath, -a         Use absolute paths in output.
```

7.2 File modelines

You can set options for Pylava inside a source file. Use pylava *modeline* for this.

Format:

```
# pylava:{name1}={value1}:{name2}={value2}:...
```

Example:

```
.. Somethere in code
# pylava:ignore=W:select=W301
```

Disable code checking for current file:

```
.. Somethere in code
# pylava:skip=1
```

Those options have a higher priority.

7.3 Skip lines (noqa)

Just add *# noqa* in end of line to ignore.

Example:

```
def urgent_fuction():
    unused_var = 'No errors here' # noqa
```

7.4 Configuration file

Pylava looks for a configuration file in the current directory.

The program searches for the first matching ini-style configuration file in the directories of command line argument. Pylava looks for the configuration in this order:

```
pylava.ini
setup.cfg
tox.ini
pytest.ini
```

The `--option` / `-o` argument can be used to specify a configuration file.

Pylava searches for sections whose names start with *pylava*.

The *pylava* section configures global options like *linters* and *skip*.

Example:

```
[pylava]
format = pylint
skip = */.tox/*,*/.env/*
linters = pylint,mccabe
ignore = F0401,C0111,E731
```

7.5 Set Code-checkers' options

You could set options for special code checker with pylava configurations.

Example:

```
[pylava:pyflakes]
builtins = _

[pylava:pycodestyle]
max_line_length = 100

[pylava:pylint]
max_line_length = 100
disable = R
```

See code-checkers' documentation for more info.

7.6 Set options for file (group of files)

You could set options for special file (group of files) with sections:

The options have a higher priority than in the *pylava* section.

Example:

```
[pylava:*/pylava/main.py]
ignore = C901,R0914,W0212
select = R

[pylava:*/tests.py]
ignore = C0110

[pylava:*/setup.py]
skip = 1
```

CHAPTER 8

Pytest integration

Pylava has Pytest support. The package automatically registers itself as a pytest plugin during installation. Pylava also supports *pytest_cache* plugin.

Check files with pylava:

```
pytest --pylava ...
```

Recommended way to set pylava options when using pytest — configuration files (see below).

CHAPTER 9

Writing a linter

You can write a custom extension for Pylava. Custom linter should be a python module. Name should be like `pylava_<name>`.

In `setup.py`, `pylava.linter` entry point should be defined.

Example:

```
setup(
    # ...
    entry_points={
        'pylava.linter': ['lintername = pylava_lintername.main:Linter'],
    }
    # ...
)
```

`Linter` should be instance of `pylava.lint.Linter` class. Must implement two methods:

- `allow` takes a path and returns true if linter can check this file for errors.
- `run` takes a path and meta keywords params and returns a list of errors.

9.1 Example

Just a virtual 'WOW' checker.

setup.py:

```
setup(
    name='pylava_wow',
    install_requires=[ 'setuptools' ],
    entry_points={
        'pylava.linter': ['wow = pylava_wow.main:Linter'],
    }
```

(continues on next page)

(continued from previous page)

```
    # ...
)
```

pylava_wow.py:

```
from pylava.lint import Linter as BaseLinter

class Linter(BaseLinter):

    def allow(self, path):
        return 'wow' in path

    def run(self, path, **meta):
        with open(path) as f:
            if 'wow' in f.read():
                return [{
                    lnum: 0,
                    col: 0,
                    text: 'Wow has been found.',
                    type: 'WOW'
                }]
```

CHAPTER 10

Run pylava from python code

```
from pylava.main import check_path, parse_options

# Use and/or modify 0 or more of the options defined as keys in the
# variable my_redefined_options below. To use defaults for any
# option, remove that key completely.
my_redefined_options = {
    'linters': ['pep257', 'pydocstyle', 'pycodestyle', 'pyflakes' ...],
    'ignore': ['D203', 'D213', 'D406', 'D407', 'D413' ...],
    'select': ['R1705' ...],
    'sort': 'F,E,W,C,D,...',
    'skip': '*__init__.py,*/test/*.py,...',
    'async': True,
    'force': True
    ...
}
# relative path of the directory in which pylama should check
my_path = '...'

options = parse_options([my_path], **my_redefined_options)
errors = check_path(options, rootdir='.')
```

CHAPTER 11

Support

To report bugs, suggest improvements, or ask questions, please create a new issue at http://github.com/pylava/pylava/issues.

CHAPTER 12

Contributing

Development of Pylava happens at the `master` branch of https://github.com/pylava/pylava.

CHAPTER 13

Contributors

See AUTHORS.

CHAPTER 14

License

This is free software. You are permitted to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of it, under the terms of the MIT License. See LICENSE.rst for the complete license.

This software is provided WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See LICENSE.rst for the complete disclaimer.

The original README from Pylama that made Pylama available under a BSD license and the original LICENSE file with the GNU LGPL license text are archived in the pylama-archive directory.