pyglint

Release 0.1.3

Jan 17, 2020

Contents

docs	
tests	
package	

Makes it easy to write custom Pylint checkers.

CHAPTER 1

Installation

```
pip install pyglint
```

CHAPTER 2

Documentation

https://pyglint.readthedocs.io/

CHAPTER 3

Contents

3.1 Overview

docs	
tests	
package	

Makes it easy to write custom Pylint checkers.

3.1.1 Installation

```
pip install pyglint
```

3.1.2 Documentation

https://pyglint.readthedocs.io/

3.2 Installation

At the command line:

```
pip install pyglint
```

3.3 Usage

1. Install pyglint.
2. Write a linter.

A checker takes a node and yields *Problem* objects.

class pyglint.**Problem**(*name*, *text*, *explanation*, *id*)
A problem found by a checker.

Parameters

- **name** (str) – The name of the problem. Usually 2-4 words, hyphenated.
- **text** – The message text for display to the user. str.format() syntax is
- **Usually one short sentence.** (*supported.*) –
- **explanation** (str) – Prose description of the problem. Usually a few sentences.

Define a *Problem* beforehand with *CheckerGroup.problem()* and reference it with *CheckerGroup.check()*.

class pyglint.**CheckerGroup**(*name*, *checkers=NOTHING*, *problems=NOTHING*, *id_prefix='E'*)
The main object for defining linters with Pyglint.

check(*node_type*)
Check for one or more pre-defined *Problem* s.

Parameters **node_type** (Type[NodeNG]) – The checker will be invoked with each instance of the given node type that pylint finds.

problem(*name*, *text*, *explanation*)
Define a reusable *Problem*.

```
import astroid

import pyglint


group = pyglint.CheckerGroup("mylinter")


BAD_NAME = group.problem(
    name="bad-name",
    text="The name '{name}' is against the guidelines.",
    explanation="It's a good idea to have a useful and descriptive name. For example,␣
↪Counter instead of ctr.",
)

IMPORT_FROM = group.problem(
    "import-from",
    text="`from ... import` is not allowed.",
    explanation="Namespaces are one honkin' great idea.",
)
```

(continues on next page)

(continued from previous page)

```
@group.check(astroid.node_classes.Name)
def find_short_names(checker, node):
    if len(node.name) < 4:
        yield pyglint.message(problem=BAD_NAME, node=node, name=node.name)


@group.check(astroid.node_classes.ImportFrom)
def find_import_from(checker, node):
    yield pyglint.message(problem=IMPORT_FROM, node=node)


def register(linter):
    """Register checkers."""
    checker = pyglint.make_pylint_checker(group)
    linter.register_checker(checker(linter))
```

3. Register it with Pylint.

```
def register(linter):
    """Register checkers."""
    checker = pyglint.make_pylint_checker(chk)
    linter.register_checker(checker(linter))
```

4. Run Pylint with it.

```
python -m pylint --load-plugins examples.mylinter examples/to-be-linted.py
```

Or enable it in your Pylint configuration file.

```
# .pylintrc
load-plugins=examples.mylinter
```

3.4 Reference

3.4.1 pyglint package

Module contents

Concise checker definition for Pylint.

class pyglint.**CheckerGroup**(*name*, *checkers=NOTHING*, *problems=NOTHING*, *id_prefix='E'*)

Bases: object

The main object for defining linters with Pyglint.

check(*node_type*)

Check for one or more pre-defined *Problem* s.

Parameters **node_type** (Type[NodeNG]) – The checker will be invoked with each instance of the given node type that pylint finds.

problem(*name*, *text*, *explanation*)

Define a reusable *Problem*.

pyglint.**make_pylint_checker**(*group*)

Return type BaseChecker

pyglint.**message**(*node*, *problem=None*, *line=None*, *col_offset=None*, *confidence=Confidence(name='UNDEFINED', description='Warning without any associated confidence level.')*, ***data*)

Return type Message

3.5 Contributing

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

3.5.1 Bug reports

When 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.

3.5.2 Documentation improvements

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

3.5.3 Feature requests and feedback

The best way to send feedback is to file an issue at https://github.com/metatooling/pyglint/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 code contributions are welcome :)

3.5.4 Development

To set up *pyglint* for local development:

1. Fork pyglint (look for the "Fork" button).
2. Clone your fork locally:

   ```
   git clone git@github.com:your_name_here/pyglint.git
   ```

3. Create a branch for local development:

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

   Now you can make your changes locally.

4. When you're done making changes, run all the checks, doc builder and spell checker with tox one command:

```
tox
```

5. 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
```

6. Submit a pull request through the GitHub website.

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 `tox`)[1].
2. Update documentation when there's new API, functionality etc.
3. Add a file in `changelog.d/` describing the changes. The filename should be `{id}.{type}.rst`, where `{id}` is the number of the GitHub issue or pull request and `{type}` is one of `breaking` (for breaking changes), `deprecation` (for deprecations), or `change` (for non-breaking changes). For example, to add a new feature requested in GitHub issue #1234, add a file called `changelog.d/1234.change.rst` describing the change.
4. Add yourself to `AUTHORS.rst`.

Tips

To run a subset of tests:

```
tox -e envname -- pytest -k test_myfeature
```

To run all the test environments in *parallel* (you need to `pip install detox`):

```
detox
```

3.6 Authors

- metatooling - https://github.com/metatooling

3.7 Changelog

3.7.1 0.1.0 (2020-01-16)

Changes

- First release on PyPI.

[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 ...

—

3.8 Indices and tables

- genindex
- modindex
- search

Python Module Index

p

Index