PyLint: The Good, The Bad, and the Ugly

Moshe Zadka – https://cobordism.com

Acknowledgement of Country

Belmont (in San Francisco Bay Area Peninsula)

Ancestral homeland of the Ramaytush Ohlone people

I live in Belmont, in the San Francisco Bay Area Peninsula. I wish to acknowledge it as the ancestral homeland of the Ramaytush Ohlone people.

Lint

Find problems

in code

without running it

Linters

Black: Can fix any problem it finds!

Flake8: Best practices + plugins

You are probably already using them

So why PyLint?

What does it give you?

0.1 The Good

Hot take: PyLint is good actually!

"PyLint can save your life" is an exaggeration, for the most part. But not as much as you might think!

PyLint can keep you from *really really* hard to find, complicated, bugs.

At worst, it will save you the time of a test run. At best, it can help you avoid complicated production mistakes.

PyLint

...is good actually.

0.1.1 Redefine function

Redefining functions

```
def test_add_small():
    # Math, am I right?
    assert 1 + 1 == 3

def test_add_large():
    assert 5 + 6 == 11

def test_add_small():
    assert 1 + 10 == 11
```

I...am embarassed to say how common this can be. Naming tests is weird: *nothing cares about the names* and there's often not a natural name.

0.1.2 Test works

Test works!

```
collected 2 items

test.py ..
2 passed
```

But, and here is the kicker, if you override a name, the testing infrastructure will happily just skip over the test!

In reality, these files can be hundreds of lines long, and the person adding the new test might not be aware of all the names. Unless someone is looking at test output carefully everything looks fine.

Worst of all, the *addition of the overriding test*, the *breakage of the overridden test*, and the *problem that results in prod* can be apart by days, months, or even your years.

0.1.3 Pylint finds it

PyLint: The Good

```
test.py:8:0: E0102: function already defined line 1
    (function-redefined)
```

But like a good friend, PyLint is there for you when the test becomes moo. Moo, like a cow's opinion: it doesn't matter.

0.2 The Bad

PyLint: The Bad

```
"""Inventory abstractions"""

import attrs

@attrs.define
class Laptop:
    """A laptop"""
    ident: str
    cpu: str
```

But, like a 90s sitcom, the more you get into PyLint, the more it becomes problematic.

This is completely reasonable code for an inventory modeling program. Yet, PyLint has opinions formed in the 90s, and is not afraid to state them as facts.

PyLint: The Bad

```
$ pylint laptop.py | sed -n '/^laptop/s/[^ ]*: //p'
R0903: Too few public methods (0/2) (too-few-public-methods)
```

OK, PyLint. You do you.

0.3 The Ugly

PyLint: The Ugly

"People will just disable the whole check if it's too picky"

PyLint issue 6987, July 3rd, 2022

Ever wanted to add your own unvetted opinion to a tool used by millions? PyLint has 12 million monthly downloads.

The attitude it takes towards adding a test with potentially many false positives is "eh".

0.4 Summary

0.4.1 Pin

Pin PyLint

Make sure to always use the same version

...until you choose to upgrade!

The first step is to *pin* the PyLint version. Your build, or CI, should not run the "latest": it should run the same version each time.

Upgrade often, but always under your control.

Pin PyLint: Loose

```
# pyproject.toml

[project.optional-dependencies]
pylint = ["pylint"]
```

Pin PyLint: Strict

```
# noxfile.py
...
@nox.session(python=VERSIONS[-1])
def refresh_deps(session):
    """Refresh the requirements-*.txt files"""
    session.install("pip-tools")
    for deps in [..., "pylint"]:
        session.run(
            "pip-compile",
            "--extra",
            deps,
```

```
            "pyproject.toml",
            "--output-file",
            f"requirements-{deps}.txt",
        )
```

...or use a service.

PyLint is fine, but you need to interact with it carefully. Pin the PyLint version you use to avoid any surprises!

0.4.2 A Few of My Favorite Things

A few of my favorite things

```
checkers = [
    "missing-class-docstring",
    "missing-function-docstring",
    "missing-module-docstring",
    "function-redefined",
]
```

These are some of the ones I like. Enforce consistency in the project, avoid some obvious mistakes

0.5 Default deny

Default Deny

```
# noxfile.py
...
@nox.session(python="3.10")
def lint(session):
    files = ["src/", "noxfile.py"]
    session.install("-r", "requirements-pylint.txt")
    session.install("-e", ".")
    session.run(
        "pylint",
        "--disable=all",
        *(f"--enable={checker}" for checker in checkers)
        "src",
    )
```

Disable all checks. Then enable ones that you think have a high value-to-false-positive ratio. (Not just false-negative-to-false-positive ratio!)

PyLint

- Keep the good: CI Checkers
- Lose the bad: Default deny

- Avoid the ugly: Pin

Take the good parts of PyLint: run it in CI to keep consistency, use the highest value checkers.

Lose the bad parts of PyLint: default deny checkers.

Avoid the ugly parts: pin the version to avoid surprises.

Unlike the facts of life, here we get to only take the good parts! And there you have the facts of PyLint.