Lint your code responsibly!

Ania Kapuścińska

#PyDays19

@lambdanis

Who am I?

- Software developer,
  currently at Genomics England
- Former astronomy educator & communicator
- Attended all PyDays Vienna!
- Sometimes uses Python in a nasty way...

@lambdanis

```
# who needs other numbers

def int(x):
    return 42

float = int
```

@lambdanis

```
MAX_INPUT_SIZE = 1000

def foo():
    global MAX_INPUT_SIZE
    MAX_INPUT_SIZE = 0


def add_field(name, fields=[]):
    fields.append(name)
    return fields
```

@lambdanis

Linter

noun

1. A machine for **removing** the short fibers from cotton seeds after ginning

2. A tool that analyzes source code **to flag** programming errors, bugs, stylistic errors (...)

@lambdanis

```
import letters
def generate_report(user, date_from, date_to, output_format, columns, metrics, filters, page_size):
    query = generate_report_query(columns,
                metrics,
                            filters)


    if output_format!='csv' and \
            output_format   !=   'txt' and output_format!='xlsx':
        rs = prepare_result(query,     page_size,     output_format)
    else:



        rs=prepare_export_file(query,output_format)

    return rs
```

@lambdanis

Pycodestyle

```
$ pycodestyle code

E501 line too long (99 > 79 characters)
E128 continuation line under-indented for visual indent
E225 missing whitespace around operator
E303 too many blank lines (3)
```

- Formerly pep8
- Checks against pep8

@lambdanis

Just use a formatter!

YAPF

“the configurable one”

“the uncompromising one”

@lambdanis

```
import letters
from report_utils import prepare_result, prepare_export_file


def generate_report(
        user, date_from, date_to, output_format, columns, metrics, filters,
        page_size
):
    query = generate_report_query(columns, metrics)

    if output_format != 'csv' and output_format != 'txt' and output_format != 'xlsx':
        rs = prepare_result(query, page_size, output_format)
    else:
        rs = prepare_export_file(query, page_size, output_format)

    return rs
```

@lambdanis

Pyflakes

```
$ pyflakes code

'letters' imported but unused
undefined name 'generate_report_query'
```

- “it will never complain about style, and it will try very, very hard to never emit false positives”
- Quick
- Checks each file separately

@lambdanis

```python
from report_utils import (
    prepare_result, prepare_export_file, generate_report_query
)


def generate_report(
        user, date_from, date_to, output_format, columns, metrics, filters,
        page_size
):
    query = generate_report_query(columns, metrics)

    if output_format != 'csv' and output_format != 'txt' and output_format != 'xlsx':
        rs = prepare_result(query, page_size, output_format)
    else:
        rs = prepare_export_file(query, page_size, output_format)

    return rs
```

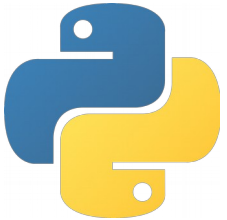
@lambdanis

```
$ pylint code

E1120: No value for argument 'filters' in function call (no-value-for-parameter)
E1121: Too many positional arguments for function call (too-many-function-args)

W0613: Unused argument 'date_from' (unused-argument)
W0613: Unused argument 'date_to' (unused-argument)

C0111: Missing function docstring (missing-docstring)
C0103: Variable name "rs" doesn't conform to snake_case naming style (invalid-name)

R0913: Too many arguments (8/5) (too-many-arguments)
R1714: Consider merging these comparisons with "in" to "output_format not in ('csv', 'txt', 'xlsx')
" (consider-using-in)
```

@lambdanis

Pylint

- The most comprehensive (or “picky”)
- Can take a while to run
- Checks dependencies in the project
- Highly configurable

Other  linters

- Pydocstyle (formerly pep257)
- Mypy – type checking
- Bandit – common security vulnerabilities

More:
https://github.com/vintasoftware/python-linters-and-code-analysis

 @lambdanis

Do it yourself!

@lambdanis

```python
class ConstantsChecker(BaseChecker):

    __implements__ = (IAstroidChecker,)

    name = 'constants'
    priority = -1
    msgs = {
        'C9999': (
            'Use a constant instead of report format literal.',
            'category-literal',
            'Report formats should be specified as constants, not literals.'
        ),
    }
    options = ()

    @utils.check_messages('category-literal')
    def visit_const(self, node):
        if node.value in REPORT_FORMATS and not (
            isinstance(node.parent, astroid.nodes.Assign)
            and len(node.parent.targets) == 1
            and getattr(node.parent.targets[0], 'name', '') in REPORT_FORMATS_CONSTANTS_NAMES
        ):
            self.add_message('category-literal', node=node)
```

@lambdanis

```
def visit_const(self, node):
    if node.value in REPORT_FORMATS and not (
        isinstance(node.parent, astroid.nodes.Assign)
        and len(node.parent.targets) == 1
        and getattr(node.parent.targets[0], 'name', '') in REPORT_FORMATS_CONSTANTS_NAMES
    ):
        self.add_message('category-literal', node=node)
```

@lambdanis

So how do I use them?

@lambdanis

Command line

```
$ pylint --load-plugins=constants_plugin code
```

@lambdanis

Continuous Integration

Travis CI

circleci

Jenkins

@lambdanis

Git pre-commit hook

Check only modified files!

```
git diff --cached --name-only --diff-filter=d | \

xargs -d '\n' --no-run-if-empty pylint
```

@lambdanis

Code editor

```
1   import letters
2   from report_utils import prepare_result, prepare_export_file
3   def generate_report(user, date_from, date_to, output_format, columns, metrics, filters, page_size):
4       query = generate_report_query(columns, metrics)
5
6
7
8       if output_format!='csv' and output_format    !=    'txt' and \
9                           output_format!=    'xlsx':
10          rs     = prepare_result(      query,    page_size,
11                                                   output_format )
12      else:
13
14
15
16          rs=prepare_export_
17
18      return rs
```

Analysis completed

1 error found
2 warnings found
9 weak warnings found

So far so good...

@lambdanis

```
def generate_report(
        user, date_from, date_to, output_format, columns, metrics, filters,
        page_size
):
    """
    Generate a report.

    :param user: user
    :param date_from: date to
    :param date_to: date from
    :param output_format: 'json' or 'csv'
    :param columns: list of columns
    :param metrics: list of columns
    :param filters: list of columns
    :param page_size: page number

    :return: report
    :rtype: dict
    """
```

@lambdanis

```
def generate_report(
        user, date_from, date_to, output_format, columns, metrics, filters,
        page_size
):
    """
    Generate a report.

    :param user: user
    :param date_from: date to
    :param date_to: date from
    :param output_format: 'json' or 'csv'
    :param columns: list of columns
    :param metrics: list of columns
    :param filters: list of columns
    :param page_size: page number

    :return: report
    :rtype: dict
    """
```

@lambdanis

Commit loop

1. git commit
2. fix linter errors
3. repeat

...

n. OMG, linters passed,
   my code is so good!!!

@lambdanis

What can go wrong?

- Code written only for linter

What can go wrong?

- Code written only for linter
- Outdated code written only for linter

What can go wrong?

- Code written only for linter
- Outdated code written only for linter
- Deceptive safety

What can go wrong?

- Code written only for linter
- Outdated code written only for linter
- Deceptive safety
- Conflicts between different linters

@lambdanis

What can go wrong?

- Code written only for linter
- Outdated code written only for linter
- Deceptive safety
- Conflicts between different linters
- Too long runs

@lambdanis

Don’t obey all the rules.

Configure your linters!

@lambdanis

pylintrc

```
[MASTER]
load-plugins =
    pylint.extensions.docparams,
    constants_plugin

[MESSAGES CONTROL]
disable =
    W9012
    R0913
    R0801
    C0103

[FORMAT]
max-line-length = 100
```

@lambdanis

```
R0913: Too many arguments (8/5) (too-many-arguments)
```

@lambdanis

pylintrc

```
[DESIGN]

max-args=10
```

@lambdanis

pylintrc

```
[DESIGN]

max-args=10
```

```python
def generate_report(
        user, date_from, date_to, output_format, columns, metrics, filters,
        page_size
):  # pylint: disable=R0913
```

@lambdanis

Don’t overuse it!

```
from code.report_utils import (
    prepare_result, prepare_export_file, generate_report_query
)  # noqa


def generate_report(  # pylint: disable=R0913
        user, date_from, date_to, output_format, columns, metrics, filters,  # noqa
        page_size  # noqa
):  # pylint: disable=R0913
    query = generate_report_query(columns, metrics)  # pylint: disable=C0301

    if output_format not in ['csv', 'txt', 'xlsx']:  # pylint: disable=C0103
        result = prepare_result(query, page_size, output_format)  # pylint: disable=R1714
    else:  # noqa
        result = prepare_export_file(query, page_size, output_format)  # pylint: disable=R1714

    return result  # noqa  # pylint: disable=C0301,C0103
```

So what can I ignore?

Will a linter help me at all?

@lambdanis

If you have...

- Poor tests coverage
- Not using a formatter
- Lots of refactoring
- Developers not familiar with best practices or conventions

If you have...

- Poor tests coverage
- Not using a formatter
- Lots of refactoring
- Developers not familiar with best practices or conventions

- Use a linter
- But don’t stop there!

@lambdanis

If you have...

- Good tests coverage
- Using formatter
- Everyone following conventions and best practices

@lambdanis

If you have...

- Good tests coverage
- Using formatter
- Everyone following conventions and best practices

- Also use a linter
- Finding bugs quicker
- Finding unused code, etc.

@lambdanis

Please lint your code responsibly :)

Thank you!

@lambdanis