

Linters and formatters

Основные понятия

Linters (статический анализ кода)

Линтеры анализируют код для обнаружения потенциальных багов, стилистических ошибок и подозрительных конструкций. Они помогают разработчикам избежать распространенных недочетов и ошибок.

Formatters (автоматическое форматирование кода)

Автоматически улучшают согласованность, читаемость, понятность кода. Помогают придерживаться определенного стиля кодирования не тратя много времени на ручную правку кода.



Основные понятия

Типы линтеров:

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

Типы форматтеров:

<https://github.com/life4/awesome-python-code-formatters>

Основные понятия

Почему следует использовать линтеры?

- Сделают ваш код чище.
- Сделают код более удобным для поддержки и чтения.
- Позволяют своевременно выявлять ошибки и недочеты в коде.
- Помогают облегчить совместную работу над проектом.
- Улучшают продуктивность при написании кода.
- Помогают в обучении писать чистый код.

Почему не следует использовать линтеры?

- Чрезмерная зависимость приводит к ложному чувству безопасности.
- Может привести к значительным накладным расходам при выполнении небольших и простых проектов.
- Может потребоваться много времени на обучение для тех, кто только начинает осваивать Python.

Основные понятия

Когда использовать линтеры?

- Во время разработки.
- На этапе коммита кода (pre-commit).
- На этапе GitHub CI.
- Во время code review.

PEP8

PEP8 — документ, который описывает соглашение о том, как писать код для языка Python, включая стандартную библиотеку, входящую в состав Python.

<https://peps.python.org/pep-0008/>

Гвидо Ван Россум: «Код читается гораздо чаще, чем пишется»

Некоторые соглашения о кодировании в PEP8:

- Предпочтительным методом отступа являются пробелы.
- Используйте 4 пробела на каждый уровень отступа.
- Длина строк максимум 79 символов.
- Разделяйте определения функций и классов верхнего уровня двумя пустыми строками.
- Методы внутри класса отделяются одной пустой строкой.
- Импорты следует сгруппировать в следующем порядке:
 - Импорты стандартной библиотеки.
 - Импорты сторонних библиотек.
 - Импорты модулей текущего проекта.
 - Пустая строка между каждой группой импортов.

pycodestyle

Pycodestyle (ранее PEP8) — официальный инструмент (линтер) для проверки кода Python на соответствие соглашениям по стилю PEP8 Python.

Установка:

```
pip install pycodestyle
```

Использование:

```
pycodestyle pycodestyle_example.py
```

```
python -m pycodestyle pycodestyle_example.py
```

pycodestyle

```
(.venv) D:\PycharmProjects\linter_examples>pycodestyle pycodestyle_example.py
pycodestyle_example.py:2:10: E401 multiple imports on one line
pycodestyle_example.py:6:1: E302 expected 2 blank lines, found 1
pycodestyle_example.py:6:31: E203 whitespace before ':'
pycodestyle_example.py:9:25: E231 missing whitespace after ','
pycodestyle_example.py:16:21: E701 multiple statements on one line (colon)
pycodestyle_example.py:17:13: E271 multiple spaces after keyword
pycodestyle_example.py:17:14: E203 whitespace before ':'
pycodestyle_example.py:17:15: E701 multiple statements on one line (colon)
pycodestyle_example.py:17:29: W292 no newline at end of file
```

Общая сводка: `pycodestyle --statistics -qq pycodestyle_example.py`

```
2      E203 whitespace before ':'
1      E231 missing whitespace after ','
1      E271 multiple spaces after keyword
1      E302 expected 2 blank lines, found 1
1      E401 multiple imports on one line
2      E701 multiple statements on one line (colon)
1      W292 no newline at end of file
```

```
from __future__ import print_function
import os, sys
import logging
from .. import views
```

```
class DoSomething (SomeCommand) :
```

```
    def __init__(self):
        for i in range(1,11):
            if self.number == i:
                print("matched")
            else:
                print('not matched')
```

```
    def check_user(self):
        if self.user: return True
        else : return False
```

Показать ошибку и описание того, как ее устранить: `pycodestyle --show-source --show-pep8 pycodestyle_example.py`

```
pycodestyle_example.py:2:10: E401 multiple imports on one line
import os, sys
    ^
```

Place imports on separate lines.

Okay: `import os\nimport sys`

E401: `import sys, os`

pycodestyle

Также можно задать файл конфигурации: tox.ini или setup.cfg

```
[pycodestyle]
ignore = E501, W291
max-line-length = 88
statistics = True
```

PEP8 Error/Warning code

Error/ Warning	Meaning
Starting with E...	Errors
Starting with W...	Warnings
100 type ...	Indentation
200 type ...	Whitespace
300 type ...	Blank lines
400 type ...	Imports
500 type ...	Line length
600 type ...	Deprecation
700 type ...	Statements
900 type ...	Syntax errors

Pylint

Pylint — универсальный линтер Python для статического анализа кода. Он проверяет код Python на соответствие широкому спектру стандартов программирования, выделяет ошибки, может искать code smells и предлагать варианты по рефакторингу кода.

Установка:

```
pip install pylint
```

Использование:

```
pylint pylint_example.py
```

```
python -m pylint pylint_example.py
```

```
pylint your_package/
```

Pylint

```
pylint_example.py:17:0: C0304: Final newline missing (missing-final-newline)
pylint_example.py:1:0: C0114: Missing module docstring (missing-module-docstring)
pylint_example.py:2:0: C0410: Multiple imports on one line (os, sys) (multiple-imports)
pylint_example.py:4:0: E0402: Attempted relative import beyond top-level package (relative-beyond-top-level)
pylint_example.py:6:0: C0115: Missing class docstring (missing-class-docstring)
pylint_example.py:6:18: E0602: Undefined variable 'SomeCommand' (undefined-variable)
pylint_example.py:15:4: C0116: Missing function or method docstring (missing-function-docstring)
pylint_example.py:16:8: R1703: The if statement can be replaced with 'return bool(test)' (simplifiable-if-statement)
pylint_example.py:16:8: R1705: Unnecessary "else" after "return", remove the "else" and de-indent the code inside it (no-else-return)
pylint_example.py:16:22: C0321: More than one statement on a single line (multiple-statements)
pylint_example.py:6:0: R0903: Too few public methods (1/2) (too-few-public-methods)
pylint_example.py:2:0: W0611: Unused import os (unused-import)
pylint_example.py:2:0: W0611: Unused import sys (unused-import)
pylint_example.py:3:0: W0611: Unused import logging (unused-import)
pylint_example.py:4:0: W0611: Unused import views (unused-import)
```

Отображает разные типы сообщений:

https://pylint.readthedocs.io/en/stable/user_guide/messages/messages_overview.html

Можно настраивать с помощью конфигурационного файла: `.pylintrc`

```
1 [MASTER]
2 disable=
3     C0111, # missing-docstring
4     C0103 # invalid-name
5
6 [MESSAGES CONTROL]
7 max-line-length=100
```

Pylint

```
1 class my_class:
2     def func1(self):
3         pass
4
5     def anotherFunction(self, arg1):
6         self.myvar = arg1
7         print(arg1)
8
9 obj = my_class()
10 obj.func1()
11 obj.anotherFunction(10)
```



```
1 class MyClass:
2     """Example class demonstrating Pylint improvements."""
3
4     def __init__(self):
5         """Initialize the class."""
6         self.my_var = None
7
8     def function_one(self):
9         """Example method that does nothing."""
10        # Previously had 'pass', removed as it's unnecessary here.
11
12    def another_function(self, arg1):
13        """Print the provided argument.
14
15        Args:
16            arg1 (int): The argument to be printed.
17        """
18        self.my_var = arg1
19        print(arg1)
20
21 obj = MyClass()
22 obj.function_one()
23 obj.another_function(10)
```

```
***** Module pylint_example
pylint_example.py:11:0: C0304: Final newline missing (missing-final-newline)
pylint_example.py:1:0: C0114: Missing module docstring (missing-module-docstring)
pylint_example.py:1:0: C0115: Missing class docstring (missing-class-docstring)
pylint_example.py:1:0: C0103: Class name "my_class" doesn't conform to PascalCase naming style (invalid-name)
pylint_example.py:2:4: C0116: Missing function or method docstring (missing-function-docstring)
pylint_example.py:5:4: C0116: Missing function or method docstring (missing-function-docstring)
pylint_example.py:5:4: C0103: Method name "anotherFunction" doesn't conform to snake_case naming style (invalid-name)
pylint_example.py:6:8: W0201: Attribute "myvar" defined outside __init__ (attribute-defined-outside-init)
```

flake8

Оболочка, которая использует несколько анализаторов кода, в том числе и pycodestyle.

```
pylint_example.py:2:1: F401 'os' imported but unused
pylint_example.py:2:1: F401 'sys' imported but unused
pylint_example.py:2:10: E401 multiple imports on one line
pylint_example.py:3:1: F401 'logging' imported but unused
pylint_example.py:4:1: F401 '..views' imported but unused
pylint_example.py:6:1: E302 expected 2 blank lines, found 1
pylint_example.py:6:19: F821 undefined name 'SomeCommand'
pylint_example.py:6:31: E203 whitespace before ':'
pylint_example.py:9:25: E231 missing whitespace after ','
pylint_example.py:16:21: E701 multiple statements on one line (colon)
pylint_example.py:17:13: E271 multiple spaces after keyword
pylint_example.py:17:14: E203 whitespace before ':'
pylint_example.py:17:15: E701 multiple statements on one line (colon)
pylint_example.py:17:29: W292 no newline at end of file
```

Описание сообщения с примером: <https://www.flake8rules.com/rules/E701.html>

isort

isort — это утилита/библиотека Python для сортировки импорта в алфавитном порядке и автоматического разделения на разделы и по типу.

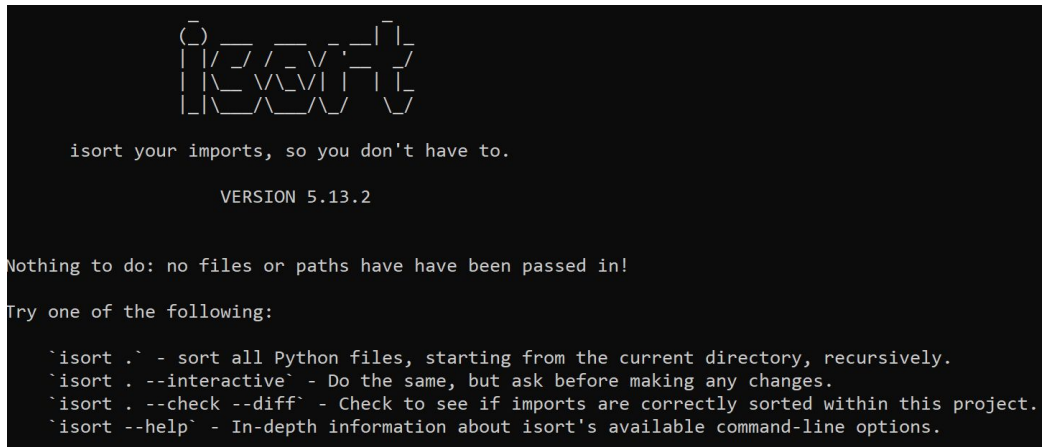
Установка:

```
pip install isort
```

Использование:

```
isort isort_example.py
```

```
python -m isort isort_example.py
```

A terminal window with a dark background showing the help text for the isort command. At the top is a logo made of ASCII characters. Below it is the tagline 'isort your imports, so you don't have to.' followed by the version number 'VERSION 5.13.2'. The main message says 'Nothing to do: no files or paths have have been passed in!' and 'Try one of the following:'. A list of commands follows: 'isort .' for recursive sorting, 'isort --interactive' for interactive mode, 'isort --check --diff' for checking and diffing, and 'isort --help' for more information.

```
(̣)
| | /  /  /  /  /  /  /  /
| | \  \  \  \  \  \  \  \
| |  \  \  \  \  \  \  \  \
| |   \  \  \  \  \  \  \  \

isort your imports, so you don't have to.

VERSION 5.13.2

Nothing to do: no files or paths have have been passed in!

Try one of the following:

`isort .` - sort all Python files, starting from the current directory, recursively.
`isort . --interactive` - Do the same, but ask before making any changes.
`isort . --check --diff` - Check to see if imports are correctly sorted within this project.
`isort --help` - In-depth information about isort's available command-line options.
```

isort

Проверка предлагаемых изменений

```
isort isort_example.py --diff
```

```
@@ -1,9 +1,10 @@
+import os
+
+import matplotlib.pyplot as plt
+import numpy as np
+import sklearn.metrics as mt
+import sklearn.model_selection as ms
+import sklearn.neural_network as nn
+import sklearn.preprocessing as pp
+    from scipy import signal
+    from scipy.io import wavfile
-import sklearn.neural_network as nn
-import sklearn.model_selection as ms
-import sklearn.preprocessing as pp
-import sklearn.metrics as mt
-import matplotlib.pyplot as plt
-import os
-import numpy as np
```

```
isort isort_example.py --check-only
```

ERROR:

D:\PycharmProjects\linter_examples\isort_example.py Imports are incorrectly sorted and/or formatted.

isort

Из PEP8:

Каждый импорт, как правило, должен быть на отдельной строке.

```
import os
import sys
```

Импорты всегда помещаются в начале файла, сразу после комментариев к модулю и строк документации, и перед объявлением констант.

Импорты должны быть сгруппированы в следующем порядке:

- импорты из стандартной библиотеки
- импорты сторонних библиотек
- импорты модулей текущего проекта

Вставляйте пустую строку между каждой группой импортов.

Рекомендуется абсолютное импортирование.

```
import mypkg.sibling
from mypkg import sibling
from mypkg.sibling import example
```

wildcard-импорты (from <module> import *) следует избегать, так как они делают неясным то, какие имена присутствуют в глобальном пространстве имён, что вводит в заблуждение как читателей, так и многие автоматизированные средства.

isort

```
isort isort_example.py
```

```
from scipy import signal
from scipy.io import wavfile
import sklearn.neural_network as nn
import sklearn.model_selection as ms
import sklearn.preprocessing as pp
import sklearn.metrics as mt
import matplotlib.pyplot as plt
import os
import numpy as np
```

```
import os

import matplotlib.pyplot as plt
import numpy as np
import sklearn.metrics as mt
import sklearn.model_selection as ms
import sklearn.neural_network as nn
import sklearn.preprocessing as pp
from scipy import signal
from scipy.io import wavfile
```

Black – “Бескомпромиссный форматировщик кода”

Black — это форматировщик кода Python, известный своим бескомпромиссным подходом к стилю кода. Он отдает приоритет единообразному стилю вне зависимости от того, в каких проектах применяется. Стиль кодирования, используемый Black, можно рассматривать как строгое подмножество PEP 8.

https://black.readthedocs.io/en/stable/the_black_code_style/current_style.html

Установка:

```
pip install black
```

```
pip install "black[jupyter]"
```



Использование:

```
black black_example.py
```

```
python -m black black_example.py
```

```
black .
```

black online: [ссылка](#)

black – “Бескомпромиссный форматировщик кода”

Параметры конфигурации минимальны:

Настройка длины строки (по умолчанию 88): `black your_script.py -l 100`

Исключить каталог или файл: `black your_directory --exclude='/migrations/'`

Можно задать файл конфигурации: `pyproject.toml`

```
1  [tool.black]
2  line-length = 100
3  exclude = '''
4  /(
5      migrations
6  )/
7  '''
```


black – “Бескомпромиссный форматировщик кода”

```
from __future__ import print_function
import os, sys
import logging
from .. import views

class DoSomething(SomeCommand) :

    def __init__(self):
        for i in range(1,11):
            if self.number == i:
                print("matched")
            else:
                print('not matched')

    def check_user(self):
        if self.user: return True
        else : return False
```




```
from __future__ import print_function
import os, sys
import logging
from .. import views

class DoSomething (SomeCommand):

    def __init__(self):
        for i in range(1, 11):
            if self.number == i:
                print("matched")
            else:
                print("not matched")

    def check_user(self):
        if self.user:
            return True
        else:
            return False
```



```
from __future__ import print_function

import logging
import os
import sys

from .. import views

class DoSomething (SomeCommand):

    def __init__(self):
        for i in range(1, 11):
            if self.number == i:
                print("matched")
            else:
                print("not matched")

    def check_user (self):
        if self.user:
            return True
        else:
            return False
```

(.venv) D:\PycharmProjects\linter_examples>black black_example.py
All done! 🌟📁🌟
1 file left unchanged.

MyPy

MyPy — это линтер для проверки типов Python, который помогает выявлять несоответствия типов в коде. Применяя подсказки типов, MyPy гарантирует, что функции и переменные используются правильно на основе их типов данных, что снижает количество ошибок во время выполнения.

```
from typing import List, Dict

def square_numbers(numbers: List[int]) -> Dict[int, int]:
    return {number: number**2 for number in numbers}

result = square_numbers((1, 2, 3, 4))
print(result)
```

```
mypy_example.py:6: error: Argument 1 to "square_numbers" has incompatible type "tuple[int, int, int, int]"; expected "list[int]" [arg-type]
Found 1 error in 1 file (checked 1 source file)
```

Ruff



Быстрый линтер и форматировщик кода на Python, написанный на Rust.

```
from __future__ import print_function
import os, sys
import logging
from .. import views
```

```
class DoSomething(SomeCommand):
    def __init__(self):
        for i in range(1, 11):
            if self.number == i:
                print("matched")
            else:
                print("not matched")

    def check_user(self):
        if self.user:
            return True
        else:
            return False
```

```
(.venv) D:\PycharmProjects\linter_examples>ruff check ruff_example.py
ruff_example.py:2:1: E401 [*] Multiple imports on one line
|
|   from __future__ import print_function
|   import os, sys
|   ^^^^^^^^^^^^^^^^^ E401
|   import logging
|   from .. import views
|
| = help: Split imports

ruff_example.py:2:8: F401 [*] `os` imported but unused
|
|   from __future__ import print_function
|   import os, sys
|         ^^ F401
|   import logging
|   from .. import views
|
| = help: Remove unused import

ruff_example.py:2:12: F401 [*] `sys` imported but unused
|
|   from __future__ import print_function
|   import os, sys
```

pre-commit

Pre-commit hooks — это важный инструмент для поддержания качества и согласованности кода. Это автоматизированные скрипты, которые выполняют проверки перед фиксацией в репозитории, гарантируя, что все изменения соответствуют требуемым стандартам.

Установка: `pip install pre-commit`

Файл конфигурации: `.pre-commit-config.yaml`

Установка hooks: `pre-commit install`

```
1 repos:
2   - repo: https://github.com/psf/black
3     rev: stable
4     hooks:
5       - id: black
6
7   - repo: https://github.com/PyCQA/pylint
8     rev: pylint-2.9.6
9     hooks:
10      - id: pylint
11
12   - repo: https://github.com/pre-commit/mirrors-mypy
13     rev: v0.910
14     hooks:
15       - id: mypy
16
17   - repo: https://github.com/PyCQA/bandit
18     rev: 1.7.0
19     hooks:
20       - id: bandit
```