2.2. Исследовать способы преобразования программного кода Python в соответствии со стандартом PEP8.

Автоматическое обнаружение инструмента Pylint

**Pylint - это библиотека, которая проверяет нарушения спецификации PEP8 и распространенные ошибки**, Он автоматически найдет код, который не соответствует стандартам стиля кода и потенциальным проблемам, и выведет соответствующую информацию о нарушении спецификации и проблеме в коде вывода консоли.

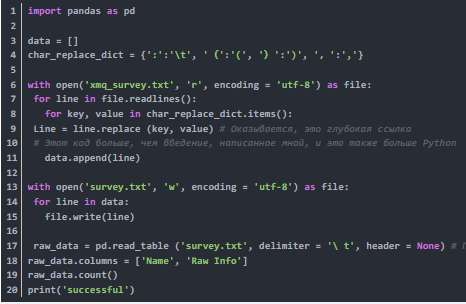
**1. Установка и использование**

Как и в других библиотеках Python, вы можете напрямую установить pylint для завершения установки. Кроме того, Anaconda поставляется с Pylint, поэтому если вы установили Anaconda, вам не нужно устанавливать эту библиотеку отдельно.

Использование pylint также очень простое. Основное использование - это прямой ввод пути / имени модуля pylint в консоли для проверки спецификаций стиля кода связанных модулей, и результаты проверки будут выводиться на консоль.

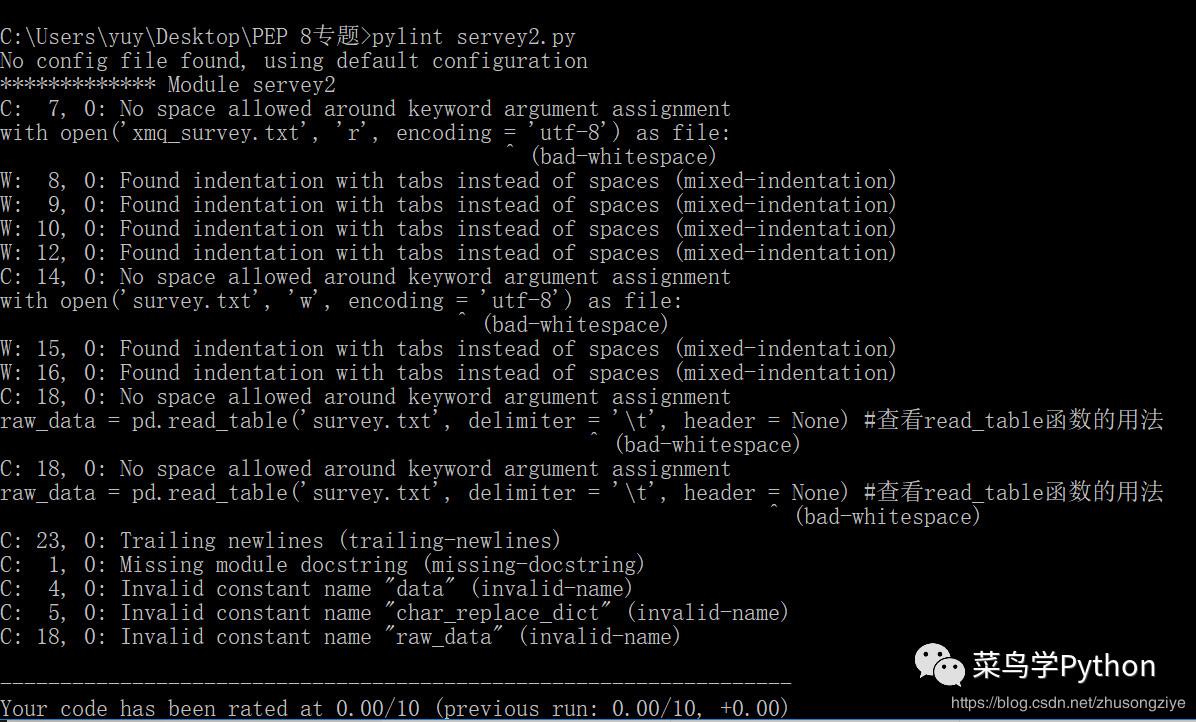
Рекомендуется совмещать подсказки pylint --help для обучения и поиска. После ознакомления с основной ситуацией с Pylint давайте сделаем подробное объяснение на примере.

**2. Пример демонстрации**



На первый взгляд кажется, что большой проблемы нет, но после осмотра пилона он дает несколько советов по проблемам (ниже), давайте посмотрим на результаты проверки

**Каждая строка информации, начинающаяся с заглавной буквы + двоеточия, представляет собой запрос обратной связи**。



Заглавная буква в начале указывает на тип ошибки (**В основном это C \ R \ W \ E \ F**); Два числа, разделенные запятыми, указывают место (строку и столбец), в котором была обнаружена проблема, после чего следует подробное описание проблемы. Содержимое в скобках называется идентификатором сообщения, который можно просто понять как подробную классификацию типа ошибки.

Команда pylint --help-msg = <msg-id> может просмотреть подробную информацию об этой проблеме.

C-нарушение стандартов стиля кода;

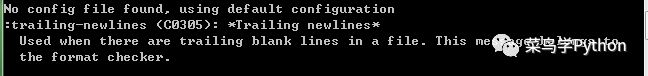
R-Плохая структура кода;

W-предупреждение о деталях;

E-есть ошибка в коде;

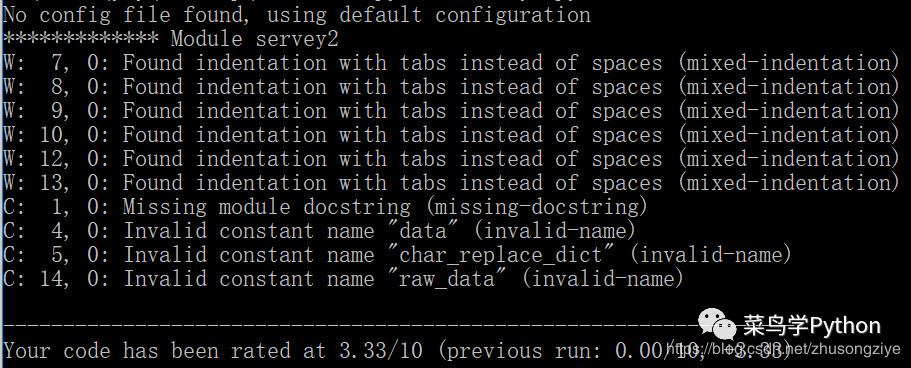
F-Ошибка, из-за которой Pylint перестал работать.

Например, если мы выполним команду pylint --help-msg = trailing-newlines, в консоли будет выведено подробное описание проблемы trailing-newlines:



Последняя строка вывода pylint - это его оценка для этого теста. Полная оценка составляет 10 баллов. Вы можете видеть, что мой код на этот раз оценивается в 0 баллов.

Теперь, когда вы знаете, где ваш код не стандартизирован, продолжайте и исправьте его. В соответствии с подсказкой сначала присвойте функции параметры = пробелы с обеих сторон, затем удалите лишние пустые строки, а затем запустите pylint для повторного обнаружения и получите следующее Результат:



Как видите, больше нет подсказок о проблемах с только что измененным кодом, **Счет также увеличился с 0 до 3,33**, Но есть еще много проблем,**Здесь следует отметить, что PEP8 не должен соблюдаться на 100%. Если следование спецификации PEP 8 сделает код менее читаемым и несовместимым со стилем окружающего кода, вам все равно придется следовать собственному суждению.**

В этом случае Pylint также предоставляет операцию, которая может вручную блокировать некоторые проблемные запросы.Принимая предыдущий код в качестве примера, оставшиеся несколько проблем в основном связаны с использованием клавиши Tab, именования переменных и отсутствием документации. Вызвано, мы можем использовать

Команда pylint --disable = mixex-indentation, invalid-name, missing-docstring name name для маскировки и повторной проверки соответствующих спецификаций, обнаружила, что все проблемные подсказки были устранены, и оценка также была увеличена до 10 баллов.

**Автоматическая оптимизация инструмента Black**

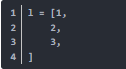
**Среди многих инструментов форматирования кода, Black является относительно новым**

Его самая большая особенность в том, что в нем меньше настраиваемых элементов, лично я думаю, что это хорошо для новичков, потому что нам не нужно слишком много думать о том, как настроить черных, и позволить черным самим принимать решение.

**1). Установка и использование**

Как и в случае с pylint, установка этого модуля может быть завершена напрямую с помощью pip install black, но черный цвет зависит от Python 3.6+, но он все еще может форматировать код Python2.

С точки зрения использования, черный читает указанный файл Python по умолчанию и форматирует его в спецификации кода, а затем выводит его в исходный файл.



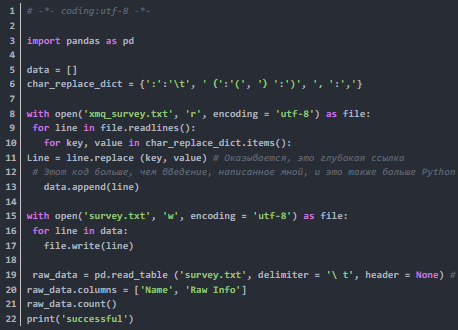
Например, мы сохраняем приведенный выше код как test.py, затем выполняем черную команду test.py в консоли, снова открываем test.py и обнаруживаем, что код в нем выглядит следующим образом:

**l = [1, 2, 3]**

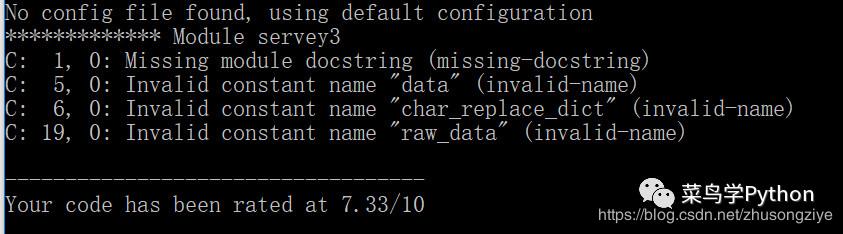
Конечно, высокая степень упаковки Black также имеет настраиваемые элементы конфигурации, такие как --version для просмотра версии, --help для просмотра справочной информации и --diff для вывода измененной информации на консоль без изменения исходного файла. Мы все еще объединяем пример для демонстрации.

### ****2). Пример демонстрации****

Здесь мы все еще используем часть кода pylint для демонстрации. Благодаря вышеописанной операции мы знаем, что тестирование этого кода напрямую с использованием pylint выдаст много проблемных запросов и даст оценку 0. Теперь мы сначала отформатируем его, используя black, и получим следующий код:



**Может показаться, что разница между кодом до и после модификации не очень очевидна: по сути, black заменил и изменил назначение параметров в коде = пробелы на обоих концах, формат комментария и символ табуляции.** Мы используем pylint для проверки, выполняем команду имени модуля pylint и получаем следующий результат:



**Видно, что по сравнению с исходным файлом оценка увеличена с 0 до 7,3, а вывод запроса вопроса значительно меньше.** Остальные проблемы связаны, в основном, с отсутствием документации и нерегулярным именованием переменных.

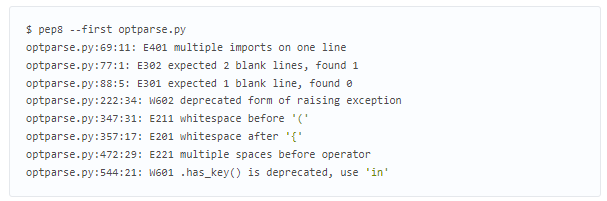
Автоматизируем форматирование

Если посмотреть сколько всяких правил в pep8, то можно сесть за рефакторинг надолго. Вот только это лениво, да и при написании нового кода сиравно будут какие-то ошибки правил. Для этого рассмотрим как же себе можно упростить жизнь.

pep8

Дабы иметь представление сколько ошибок оформления в коде, стоит использовать утилиту pep8. У нее достаточный список параметров, который позволяет рекурсивно просмотреть все файлы в папках на предмет соответствия стандарту pep8.

Вывод утилиты примерно такой:



По нему можно однозначно понять: где ошибка и что случилось.

autopep8

Ошибки стандарта часто повторяются от файла в файлу. И возникает сильное желание исправление автоматизировать. В этом случае на арену выходит autopep8. Как и pep8, он умеет самостоятельно определять ошибки, а также исправлять их. Список исправляемых ошибок форматирования можно найти здесь

Само использование autopep8 крайне простое и может выглядеть так:

$ autopep8 ./ --recursive --in-place -a

После выполнения данной команды, утилита рекурсивно пойдет по подпапкам и начнет в самих же файлах исправлять ошибки.

autoflake

Можно пойти дальше и в качестве оружия взять autoflake. Эта утилита помогает удалить не используемые импорты и переменные.

Используется примерно так:

$ autoflake --in-place --remove-all-unused-imports --remove-unused-variables -r ./

Тем самым будут рекурсивно почищены файлы в директории.

unify

Крайний, заключительный момент в редактировании кода — это строки. Кто-то любит их писать в одиночных апострофах, кто-то в двойных. Вот только и для этого существует рекомендации, а также и утилита, которая позволяет автоматически приводить в соответствие — unify

Использование:

$ unify --in-place -r ./src/

Как и везде, утилита выполнит свое грязное дело рекурсивно для файлов в папке.

docformatter

Все время говорим о самом коде, а о комментариях еще ни разу не шло речи. Настало время — docformatter. Эта утилита помогает привести ваши docstring по соглашению PEP 257. Соглашение предписывает как следует оформлять документацию.

Использование утилиты ничуть не сложнее предыдущих:

$ docformatter --in-place example.py