Жил-был чудный веб-интерфейса и все у него было хорошо: в него ходили пользователи, что-то там кликали, переходили по ссылкам, получали результат. Но со временем некоторые его странички стали тупить и долго грузиться. Менеджеры часто жалуются, мол "вот тут список долго грузился" или "интерфейсик тупит, поиск не работает". Но так трудно отделить те случаи, где проблемы на их стороне, а где действительно виноват веб-сервис. В логи интерфейса добавили время запроса ( $request\_time в nginx http://nginx.org/en/docs/http/ngx\_http\_log\_module.html#log\_format). Теперь можно распарсить логи и провести первичный анализ, выявив подозрительные URL'ы.

Про логи: семпл лога: nginx-access-ui.log-20170630.gz шаблон названия логов интерфейса соответствует названию сэмпла (ну, только время меняется) так вышло, что логи могут быть и plain и gzip лог ротируется раз в день опять же, так вышло, что логи интерфейса лежат в папке с логами других сервисов

Про отчет:

count - сколько раз встречается URL, абсолютное значение

count\_perc - сколько раз встречается URL, в процентнах относительно общего числа запросов

time\_sum - суммарный $request\_time для данного URL'а, абсолютное значение

time\_perc - суммарный $request\_time для данного URL'а, в процентах относительно общего $request\_time всех запросов

time\_avg - средний $request\_time для данного URL'а

time\_max - максимальный $request\_time для данного URL'а

time\_med - медиана $request\_time для данного URL'а

Задание: реализовать анализатор логов log\_analyzer.py .

Основная функциональность:

1. Скрипт обрабатывает при запуске последний (со самой свежей датой в имени, не по mtime файла!) лог в LOG\_DIR , в результате работы должен получится отчет как в report-2017.06.30.html (для корректной работы нужно будет найти и принести себе на диск jquery.tablesorter.min.js ). То есть скрипт читает лог, парсит нужные поля, считает необходимую статистику по url'ам и рендерит шаблон report.html (в шаблоне нужно только подставить $table\_json ). Ситуация, что логов на обработку нет возможна, это не должно являться ошибкой.

2. Если удачно обработал, то работу не переделывает при повторном запуске. Готовые отчеты лежат в REPORT\_DIR . В отчет попадает REPORT\_SIZE URL'ов с наибольшим суммарным временем обработки ( time\_sum ).

3. Скрипту должно быть возможно указать считать конфиг из другого файла, передав его путь через --config . У пути конфига должно быть дефолтное значение. Если файл не существует или не парсится, нужно выходить с ошибкой.

4. В переменной config находится конфиг по умолчанию (и его не надо выносить в файл). В конфиге, считанном из файла, могут быть переопределены перменные дефолтного конфига (некоторые, все или никакие, т.е. файл может быть пустой) и они имеют более высокий приоритет по сравнению с дефолтным конфигом. Таким образом, результирующий конфиг получается слиянием конфига из файла и дефолтного, с приоритетом конфига из файла.

5. Использовать конфиг как глобальную переменную запрещено, т.е. обращаться в своем функционале к нему так, как будто он глобальный - нельзя. Нужно передавать как аргумент.

6. Использовать сторонние библиотеки запрещено.

Мониторинг:

1. скрипт должен писать логи через библиотеку logging в формате '[%(asctime)s] %(levelname).1s % (message)s' c датой в виде '%Y.%m.%d %H:%M:%S' (logging.basicConfig позволит настроить это в одну строчку). file:///tmp/01\_advanced\_basics/homework/doc.html2/410/30/2019

README.md - Grip

Допускается только использование уровней info , error и exception . Путь до логфайла указывается в конфиге, если не указан, лог должен писаться в stdout (параметр filename в logging.basicConfig может принимать значение None как раз для этого).

2. все возможные "неожиданные" ошибки должны попадать в лог вместе с трейсбеком (посмотрите на logging.exception). Имеются в виду ошибки непредусмотренные логикой работы, приводящие к остановке обработки и выходу: баги, нажатие ctrl+C, кончилось место на диске и т.п.

3. должно быть предусмотрено оповещение о том, что большую часть анализируемого лога не удалось распарсить (например, потому что сменился формат логирования). Для этого нужно задаться относительным (в долях/процентах) порогом ошибок парсинга и при его превышании писать в лог, затем выходить.

Тестирование:

1. на скрипт должны быть написаны тесты с использованием библиотеки unittest

(https://pymotw.com/2/unittest/). Имя скрипта с тестами должно начинаться со слова test . Тестируемые кейсы и структура тестов определяется самостоятельно (без фанатизма, в принципе достаточно функциональных тестов).

Цель задания: получить (прокачать) навык написания production-ready кода. То есть адекватного кода, который удобно расширять и поддерживать, протестированного и пригодного для мониторинга. Совпадение всех чисел с приведенным примером отчета целью не является (лишь бы похожи были =)

Критерии успеха: задание обязательно, критерием успеха является работающий согласно заданию код, для которого написаны тесты, проверено соответствие pep8, написана минимальная документация с примерами запуска (боевого и тестов), в README, например. Далее успешность определяется code review.

Распространенные проблемы:

не стоит делать свои кастомные классы ошибок, это иногда (!) имеет смысл для библиотек, но не для задач подобного рода.

ограничьтесь уровнями логирования DEBUG, INFO и ERROR: <https://dave.cheney.net/2015/11/05/lets-talk-about-logging> не выходите через sys.exit не из main. Это затрудняет тестирование и переиспользование кода.

чтобы отрендерить шаблон не надо итерироваться по всем его строкам и искать место замены, можно воспользоваться, например, https://docs.python.org/2/library/string.html#string. T emplate.safe\_substitute .

функцию, которая будет парсить лог желательно сделать генератором.

не забывайте про кодировки, когда читаете лог и пишите отчет.

из функции, которая будет искать последний лог удобно возвращать namedtuple с указанием пути до него, распаршенной через datetime даты из имени файла и расширением, например.

распаршенная дата из имени логфайла пригодится, чтобы составить путь до отчета, это можно сделать "за один присест", не нужно проходится по всем файлам и что-то искать.

протестируйте функцию поиска лога, она не должна возвращать .bz2 файлы и т.п. Этого можно добиться правильной регуляркой.

найти самый свежий лог можно за один проход по файлам, без использования glob, сортировки и т.п.

нужный открыватель лога (open/gzip.open) перед парсингом можно выбрать через тернарный оператор.

проверка на превышение процента ошибок при парсинге выполняетя один раз, в конце чтения файла, а не на каждую строчку/ошибку.